UNCLASSIFIED

AD NUMBER
ADB120252
LIMITATION CHANGES
TO: **Approved for public release; distribution is unlimited.**
FROM: **Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; MAR 1988. Other requests shall be referred to Air Force Armament Lab., Eglin AFB, FL 32542. This document contains export-controlled technical data.**
AUTHORITY
AFSC wright lab at eglin afb, ltr 13 Feb 1992

THIS PAGE IS UNCLASSIFIED

DTIC FILE COPY

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Distribution authorized to U.S. Government Agencies and their contractors; CT (over)
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFATL-TR-88-18, Vol 5
6a. NAME OF PERFORMING ORGANIZATION: McDonnell Douglas Astronautics Company; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: Aeromechanics Division
6c. ADDRESS (City, State, and ZIP Code): P.O. Box 516, St. Louis, MO 63166	7b. ADDRESS (City, State, and ZIP Code): Air Force Armament Laboratory, Eglin AFB, FL 32542-5434
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: STARS Joint Program Office; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F08635-86-C-0025
8c. ADDRESS (City, State, and ZIP Code): Room 3D139 (1211 Fern St), The Pentagon, Washington DC 20301-3081	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 63756D; PROJECT NO. 921C; TASK NO. GZ; WORK UNIT ACCESSION NO. 57

11. TITLE (Include Security Classification)
Common Ada Missile Package (CAMP) Project: Missile Software Parts, Vol 5: Top-Level Design Document (Vol 4-6)

12. PERSONAL AUTHOR(S)
D. McNicholl, S. Cohen, C. Palmer, et al.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical Note	FROM Sep 85 TO Mar 88	March 1988	336

16. SUPPLEMENTARY NOTATION
SUBJECT TO EXPORT CONTROL LAWS.
Availability of this report is specified on verso of front cover. (over)

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Reusable Software, Missile Software, Software Generators Ada, Parts Composition, Systems, Software Parts

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The objective of the CAMP program is to demonstrate the feasibility of reusable Ada software parts in a real-time embedded application area; the domain chosen for the demonstration was that of missile flight software systems. This required that the existence of commonality within that domain be verified (in order to justify the development of parts for that domain), and that software parts be designed which address those areas identified. An associated parts system was developed to support parts usage. Volume 1 of this document is the User's Guide to the CAMP Software parts; Volume 2 is the Version Description Document; Volume 3 is the Software Product Specification; Volumes 4-6 contain the Top-Level Design Document; and, Volumes 7-12 contain the Detail Design Documents.

DTIC ELECTE APR 06 1988 E

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Christine Anderson	22b. TELEPHONE (Include Area Code): (904) 882-2961; 22c. OFFICE SYMBOL: AFATL/FXG

DD Form 1473, JUN 86 — Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

AD-B120 252

3. DISTRIBUTION/AVAILABILITY OF REPORT (CONCLUDED)

~~this report documents test and evaluation,~~ distribution limitation applied March 1988. Other requests for this document must be referred to AFATL/FXG, Eglin AFB, Florida 32542-5434.

16. SUPPLEMENTARY NOTATION (CONCLUDED)

These technical notes accompany the CAMP final report AFATL-TR-85-93 (3 Vols)

AFATL-TR-88-18, Vol. 5

SOFTWARE TOP LEVEL DESIGN DOCUMENT

FOR THE

MISSILE SOFTWARE PARTS

OF THE

COMMON ADA MISSILE PACKAGE (CAMP) PROJECT

CONTRACT F08635-86-C-0025

CDRL SEQUENCE NO. C014

30 OCTOBER 1987

Distribution authorized to U.S. Government agencies and their contractors only; CT ~~this report documents test and evaluation~~; distribution limitation applied July 1987. Other requests for this document must be referred to the Air Force Armament Laboratory (FXG) Eglin Air Force Base, Florida 32542-5434.

DESTRUCTION NOTICE – For classified documents, follow the procedures in DoD 5220.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX. For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

WARNING: This document contains technical data whose export is restricted by the Arms Export Control Act (Title 22, U.S.C., Sec. 2751, et seq.) or the Export Administration Act of 1979, as amended (Title 50, U.S.C., App. 2401, et seq.). Violations of these export laws are subject to severe criminal penalties. Disseminate in accordance with the provisions of AFR 80-34.

AIR FORCE ARMAMENT LABORATORY
Air Force Systems Command ▪ United States Air Force ▪ Eglin Air Force Base, Florida

3.6.5 KALMAN FILTER

(This page left intentionally blank.)

3.6.5.1 KALMAN_FILTER_COMMON_PARTS TLCSC (CATALOG #P159-0)

This part, which is designed as an Ada package, contains specifications for all CAMP parts which can be used to implement a Kalman Filter regardless of the type of H matrix used.

3.6.5.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this TLCSC:

Name	Requirements Allocation
State_Transition_And_Process_Noise_Matrices_Manager	R145
Error_Covariance_Matrix_Manager	R146
State_Transition_Matrix_Manger	R148

3.6.5.1.2 INPUT/OUTPUT

None.

3.6.5.1.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.5.1.4 LOCAL ENTITIES

None.

3.6.5.1.5 INTERRUPTS

None.

3.6.5.1.6 TIMING AND SEQUENCING

None.

3.6.5.1.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.5.1.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
State_Transition_And_Process_Noise_Matrices_Manager	generic package	Manages the State Transition (Phi) and Process Noise (Q) Matrices
Error_Covariance_Matrix_Manager	generic package	Manages the Error Covariance (P) Matrix
State_Transition_Matrix_Manager	generic package	Manages the State Transition (Phi) Matrix

3.6.5.1.9 PART DESIGN

3.6.5.1.9.1 STATE_TRANSITION_AND_PROCESS_NOISE_MATRICES_MANAGER (CATALOG #P160-0)

This LLCSC is a generic package which manages the State Transition (Phi) and Process Noise (Q) matrices. It consists of an Initialize procedure, a Propagation function, and functions which return the stored value of each of the two matrices.

3.6.5.1.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R145.

3.6.5.1.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Time_Intervals	floating point	Type for the delta time variable
Phi_Matrices	private	Data type of N x N Phi Matrix
Integrated_F_Matrices	private	Data type of N x N Matrix for F integration
Integrated_Q_Matrices	private	Data type of N x N Matrix for Q integration
Q_Matrices	private	Data type of N x N Q Matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Add_To_Identity	procedure	Adds the identity matrix to an Integrated_F_Matrices
Set_To_Identity_Matrix	function	Sets a Phi_Matrices type matrix to the identity matrix
Set_To_Zero_Matrix	function	Sets a Q_Matrices type matrix to the zero matrix
ABA_Transpose	function	Multiplies a Phi_Matrices type matrix by the transpose of a Q_Matrices type matrix yielding a Q_Matrices type matrix
"*"	function	Multiplies a Integrated F Matrix by a Time_Interval yielding a Integrated Q Matrix
"*"	function	Multiplies a Integrated F Matrix by a Phi Matrix yielding a Phi Matrix
"+"	function	Adds a Q Matrix to an Integrated Q Marix yielding a Q Matrix

3.6.5.1.9.1.3 LOCAL ENTITIES

Data structures:

This LLCSC stores the State Transition Matrix and the Propagated Process Noise Matrix.

3.6.5.1.9.1.4 INTERRUPTS

None.

3.6.5.1.9.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Common_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package BDT      renames Basic_Data_Types;
   package KFDT     renames Kalman_Filter_Data_Types;
   package KFCommon renames Kalman_Filter_Common_Parts;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                       (State_Indices       => State_Indices,
                        Measurement_Indices => Measurement_Indices,
                        Intervals           => BDT.Seconds);
   use KDT;
   ...
```

```
   package Phi_And_Q_Manager is new
           KFCommon.State_Transition_And_Process_Noise_Matrices
              (Time_Interval          => BDT.Seconds,
               Phi_Matrices           => KDT.
                                          N_By_N_Statically_Sparse_Matrices,
               Integrated_F_Matrices  => KDT.
                                          N_By_N_Dynamically_Sparse_Matrices,
               Integrated_Q_Matrices  => KDT.N_By_N.Diagonal_Matrices,
               Q_Matrices             => KDT.N_By_N_Symmetric.Matrices);

begin
   ...
   Phi_And_Q_Manager.Initialize;
   ...
   Phi_And_Q_Manager.Propagate (Integrated_F => My_Integrated_F,
                                Q            => My_Q,
                                Dt           => Delta_Time);
   My_Propagated_Phi => Phi_And_Q_Manager.Propagated_Phi;
   ...
```

3.6.5.1.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.5.1.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Initialize	procedure	Initializes Phi matrix to Identity matrix and Q matrix to zero matrix
Propagate	procedure	Propagates the Phi and Q matrices across time
Get_Current	procedure	Returns the current value of the Propagated Phi and Propagated Q matrices and then resets them
Propagated_Phi	function	Returns the current value of the Propagated Phi matrix

3.6.5.1.9.1.8 PART DESIGN

None.

3.6.5.1.9.2 KALMAN_FILTER_COMMON_PARTS.ERROR_COVARIANCE_MATRIX_MANAGER (CATALOG #P161-0)

This LLCSC is a generic package which manages the Error Covariance Matrix; it consists of an Initialize procedure, a Propagation procedure, and a P function, which returns the current Error Covariance matrix value

3.6.5.1.9.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R146.

3.6.5.1.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Base Type	Description
Phi_Matrices	private	Data type of N x N Phi matrix
P_And_Q_Matrices	private	Data type of N x N P_and_Q matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
ABA_Transpose	function	Multiplies a Phi matrix by the transpose of a P_and_Q matrix yielding a P_and_Q matrix
"+"	function	Adds two P_and_Q matrices yielding a P and Q matrix

3.6.5.1.9.2.3 LOCAL ENTITIES

Data structures:

The body of this LLCSC stores the Error_Covariance_Matrix.

3.6.5.1.9.2.4 INTERRUPTS

None.

3.6.5.1.9.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Common_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
```

```
   package BDT      renames Basic_Data_Types;
   package KFDT     renames Kalman_Filter_Data_Types;
   package KFCommon renames Kalman_Filter_Common_Parts;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                          (State_Indices       => State_Indices,
                           Measurement_Indices => Measurement_Indices,
                           Intervals           => BDT.Seconds);
   use KDT;
   ...
   package P_Manager is new KFCommon.Error_Covariance_Matrix_Manager
              (Phi_Matrices     => KDT.N_By_N_Symmetric.Matrices,
               P_and_Q_Matrices => KDT.
                                   N_By_N_Dynamically_Sparse_Matrices);
   ...
begin
   ...
   P_Manager.Initialize (Initial_P => My_Initial_P);
   ...
   P_Manager.Propagate (Propagated_Phi => My_Phi,
                        Propagated_Q   => My_Q);
   My_P => P_Manager.P;
   ...
```

3.6.5.1.9.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.5.1.9.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Initialize	procedure	Initializes P matrix to value supplied by the calling routine
Propagate	procedure	Propagates the P matrix using the Propagated Phi and Q matrices
P	function	Returns the current value of the P matrix

3.6.5.1.9.2.8 PART DESIGN

None.

3.6.5.1.9.3 KALMAN_FILTER_COMMON_PARTS.STATE_TRANSITION_MATRIX_MANAGER (CATALOG

This LLCSC is a generic package which manages the State Transition Matrix, commonly known as the Phi matrix. It consists of an Initialization procedure, a Propagation function, and a function which returns the stored Propagated_Phi value.

3.6.5.1.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R148.

3.6.5.1.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Base Type	Description
Integrated_F_ Matrices	private	Data type for N by N matrix for F Integration
Phi_Matrices	private	Data Type for N by N Phi matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Set_To_Identity_ Matrix	function	Sets a Phi matrix to the Identiy
"*"	function	Multiplies an Integrated F matrix by a Phi matrix yielding a Phi matrix

3.6.5.1.9.3.3 LOCAL ENTITIES

Data structures:

This package stores the Propagated Phi matrix.

3.6.5.1.9.3.4 INTERRUPTS

None.

3.6.5.1.9.3.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Common_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package BDT      renames Basic_Data_Types;
   package KFDT     renames Kalman_Filter_Data_Types;
   package KFCommon renames Kalman_Filter_Common_Parts;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                        (State_Indices       => State_Indices,
                         Measurement_Indices => Measurement_Indices,
                         Intervals           => BDT.Seconds);
   use KDT;
   ...
   package Phi_Manager is new KFCommon.State_Transition_Matrix_Manager
              (Integrated_F_Matrices
                             => KDT.N_By_N_Statically_Sparse_Matrices;
               Phi_Matrices => KDT.N_By_N_Dynamically_Sparse_Matrices);
   ...
begin
   ...
   Phi_Manager.Initialize;
   ...
   Phi_Manager.Propagate (Phi => My_Phi);
   My_Phi := Phi_Manager.Propagated_Phi;
   ...
```

3.6.5.1.9.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.5.1.9.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Initialize	procedure	Initializes Phi matrix to Identity matrix
Propagate	procedure	Propagates the Phi matrix across time
Propagated_Phi	function	Returns the current value of the Propagated Phi matrix

3.6.5.1.9.3.8 PART DESIGN

None.

(This page left intentionally blank.)

```
package Kalman_Filter_Common_Parts is

pragma PAGE;
   generic
      type Time_Intervals          is digits <>;
      type Phi_Matrices            is private;
      type Integrated_F_Matrices   is private;
      type Integrated_Q_Matrices   is private;
      type Q_Matrices              is private;
      with function Add_To_Identity
                     (Source_Matrix : Integrated_F_Matrices )
                     return Integrated_F_Matrices is <>;
      with procedure Set_To_Identity_Matrix
                     (Source : out Phi_Matrices) is <>;
      with procedure Set_To_Zero_Matrix
                     ( Source : out Q_Matrices ) is <>;
      with function Aba_Transpose
                     (A : Phi_Matrices;
                      B : Q_Matrices) return Q_Matrices is <>;
      with function "*" (Left  : Integrated_Q_Matrices;
                         Right : Time_Intervals)
                        return Integrated_Q_Matrices is <>;
      with function "*" (Left  : Integrated_F_Matrices;
                         Right : Phi_Matrices) return Phi_Matrices is <>;
      with function "+" (Left  : Integrated_Q_Matrices;
                         Right : Q_Matrices) return Q_Matrices is <>;
   package State_Transition_And_Process_Noise_Matrices_Manager is

      procedure Initialize;

      procedure Propagate( Integrated_F : in Integrated_F_Matrices;
                           Q            : in Integrated_Q_Matrices;
                           Dt           : in Time_Intervals );

      procedure Get_Current
                  ( Propagated_Phi : out Phi_Matrices;
                    Propagated_Q   : out Q_Matrices );

      function Propagated_Phi return Phi_Matrices;

   end State_Transition_And_Process_Noise_Matrices_Manager;

pragma PAGE;
   generic
      type Phi_Matrices     is private;
      type P_And_Q_Matrices is private;
      with function Aba_Transpose (A : Phi_Matrices;
                                   B : P_And_Q_Matrices )
                                 return P_And_Q_Matrices is <>;
      with function "+"( Left  : P_And_Q_Matrices;
                         Right : P_And_Q_Matrices )
                       return P_And_Q_Matrices is <>;
   package Error_Covariance_Matrix_Manager is

      procedure Initialize( Initial_P : in P_And_Q_Matrices );

      procedure Propagate (Propagated_Phi : in Phi_Matrices;
```

```
                              Propagated_Q    : in P_And_Q_Matrices );

      function P return P_And_Q_Matrices;

   end Error_Covariance_Matrix_Manager;

pragma PAGE;
   generic
      type Integrated_F_Matrices  is private;
      type Phi_Matrices           is private;
      with function "*" (Left  : Integrated_F_Matrices;
                         Right : Phi_Matrices)
                        return Phi_Matrices is <>;
      with procedure Set_To_Identity_Matrix
                        (Matrix : out Phi_Matrices) is <>;
   package State_Transition_Matrix_Manager is

      procedure Initialize;

      procedure Propagate( Integrated_F : in Integrated_F_Matrices );

      function Propagated_Phi return Phi_Matrices;

   end State_Transition_Matrix_Manager;

end Kalman_Filter_Common_Parts;
```

3.6.5.2 KALMAN_FILTER_COMPACT_H_PARTS TLCSC (CATALOG #P131-0)

This part, which is designed as an Ada package, contains specifications for all CAMP parts which can be used to implement a Kalman Filter when a compact Measurement Sensitivity Matrix (Compact H Matrix) is used

3.6.5.2.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this

Tlcsc:

Name	Requirements Allocation
Compute_Kalman_Gain	R149
Update_Error_Covariance_Matrix	R150
Update_State_Vector	R151
Sequentially_Update_Covariance_Matrix_and_ State_Vector	R152
Kalman_Update	R147
Update_Error_Covariance_Matrix_General_Form	R150

3.6.5.2.2 INPUT/OUTPUT

None.

3.6.5.2.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.5.2.4 LOCAL ENTITIES

None.

3.6.5.2.5 INTERRUPTS

None.

3.6.5.2.6 TIMING AND SEQUENCING

None.

3.6.5.2.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.5.2.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Compute_Kalman_Gain	generic function	Computes the Kalman gain vector resulting from the processing of a single component of measurement vector,Z
Update_Error_Covariance_Matrix	generic procedure	Computes the Error Covariance Matrix resulting from the processing of a single component of measurement vector,Z
Update_State_Vector	generic procedure	Computes the State Vector resulting from the processing of a single component of measurement vector,Z
Sequentially_Update_Error_Covariance_Matrix_And_State_Vector	generic package	Computes the updated Covariance Matrix, P, and state Vector, X.
Kalman_Update	generic package	Compute the updated State Vector, X, given the old X vector, the Z vector, the K vector, the Measurement Number, and the Compact H array.
Update_Error_Covariance_Matrix_General_Form	generic procedure	Computes the Error Covariance Matrix resulting from the processing of a single component of measurement vector,Z The general form of the equation is used

3.6.5.2.9 PART DESIGN

3.6.5.2.9.1 KALMAN_FILTER_COMPACT_H_PARTS.COMPUTE_KALMAN_GAIN (CATALOG #P132-0)

This unit is a generic function which computes the Kalman gain vector resulting from the processing of a single component of the measurement vector, Z.

3.6.5.2.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R149.

3.6.5.2.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table summarizes the generic formal data types required by this part:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_ Elements	floating point type	Elements contained in the Kalman Filter aggregates
P_Matrices	private	Data type of P matrix
Measurement_ Variance_Vectors	vector	Vector indexed by Measurement Indices containing Kalman_Filter_ Elements
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
Compact_H_Matrices	vector	Data type of Compact H matrix

Subprograms: The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Element	function	Extracts an element of a P Matrix

3.6.5.2.9.1.3 INTERRUPTS

None.

3.6.5.2.9.1.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Compact_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package BDT       renames Basic_Data_Types;
   package KFCompact renames Kalman_Filter_Compact_H_Parts;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
```

```
                         (State_Indices       => State_Indices,
                          Measurement_Indices => Measurement_Indices,
                          Intervals           => BDT.Seconds);
   use KDT;
   ...
   function CKG is new KFCompact.Compute_Kalman_Gain
              (State_Indices                  => State_Indices,
               Measurement_Indices            => Measurement_Indices,
               Kalman_Filter_Elements         => Kalman_Filter_Elements,
               P_Matrices                     => KDT.N_by_N_Symmetric.Matrices,
               Measurement_Variance_Vectors   => KDT.M_by_1.Vectors,
               K_Column_Vectors               => KDT.N_by_1.Vectors,
               Compact_H_Matrices             => KDT.M_by_1_Discrete_Vectors);
   ...
begin
   ...
   My_K := CKG (P                     => My_P,
                Measurement_Number    => This_Measurement,
                Compact_H             => My_Compact_H,
                Measurement_Variance  => My_Measurement_Variance);
   ...
```

3.6.5.2.9.1.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.5.2.9.1.6 DECOMPOSITION

None.

3.6.5.2.9.2 UPDATE_ERROR_COVARIANCE_MATRIX (CATALOG #P133-0)

This unit is a generic procedure which computes the updated covariance matrix resulting from the processing of a single component of the measurement vector, Z.

3.6.5.2.9.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R150.

3.6.5.2.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table summarizes the generic formal data types required by this part:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_ Elements	floating point type	Elements contained in the Kalman Filter aggregates
P_Matrices	private	Data type of P matrix
P_Row_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
Compact_H_Matrices	vector	Data type of Compact H matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Row	function	Extracts a row of a P matrix
"*"	function	A K Column Vector is multiplied by the transpose of a P Row vector, yielding a P matrix
"-"	function	Two P matrices are added, yielding a Symmetric matrix

3.6.5.2.9.2.3 INTERRUPTS

None.

3.6.5.2.9.2.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Compact_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package BDT       renames Basic_Data_Types;
   package KFCompact renames Kalman_Filter_Compact_H_Parts;
```

```
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                        (State_Indices       => State_Indices,
                         Measurement_Indices => Measurement_Indices,
                         Intervals           => BDT.Seconds);
   use KDT;
   ...
   procedure Update_P is new KFCompact.Update_Error_Covariance_Matrix
              (State_Indices              => State_Indices,
               Measurement_Indices        => Measurement_Indices,
               Kalman_Filter_Elements     => Kalman_Filter_Elements,
               P_Matrices                 => KDT.N_by_N_Symmetric.Matrices,
               P_Row_Vectors              => KDT.N_by_1.Vectors,
               K_Column_Vectors           => KDT.N_by_1.Vectors,
               Compact_H_Matrices         => KDT.M_by_1_Discrete_Vectors);
   ...
begin
   ...
   Update_P (P                    => My_P,
             Measurement_Number   => This_Measurement,
             K                    => My_K,
             Compact_H            => My_Compact_H);
   ...
```

3.6.5.2.9.2.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.5.2.9.2.6 DECOMPOSITION

None.

3.6.5.2.9.3 UPDATE_STATE_VECTOR (CATALOG #P134-0)

This unit is a generic procedure which updates the State Vector, X, given the old X vector, the Z vector, the K vector, the Measurement Number, and the Compact H array.

3.6.5.2.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R151.

3.6.5.2.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_ Elements	floating point type	Elements making up the Kalman Filter aggregates
Measurement_Vectors	vector	Vector indexed by Measurement_ Indices containing Kalman_Filter_ Elements
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
State_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
Compact_H_Matrices Vectors	vector	Vector indexed by Measurement_ Indices containing State_Indices

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
"+"	function	Add a K column vector and a state vector, yielding a state vector
"*"	function	Multiply a K column vector by a Kal- man Filter Element, yielding a K column vector

3.6.5.2.9.3.3 INTERRUPTS

None.

3.6.5.2.9.3.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Compact_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package BDT        renames Basic_Data_Types;
```

```
   package KFCompact renames Kalman_Filter_Compact_H_Parts;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                       (State_Indices       => State_Indices,
                        Measurement_Indices => Measurement_Indices,
                        Intervals           => BDT.Seconds);
   use KDT;
   ...
   package USV is new KFCompact.Update_State_Vector
             (State_Indices           => State_Indices,
              Measurement_Indices     => Measurement_Indices,
              Kalman_Filter_Elements  => Kalman_Filter_Elements,
              Measurement_Vectors     => KDT.M_by_1.Vectors,
              K_Column_Vectors        => KDT.N_by_1.Vectors,
              State_Vectors           => KDT.N_by_1.Vectors,
              Compact_H_Matrices      => KDT.M_by_1_Discrete_Vectors);
   ...
begin
   ...
   USV (X                        => My_X,
        Z                        => My_Z,
        K                        => My_K,
        Measurement_Number       => This_Measurement,
        Compact_H                => My_H);
   ...
```

3.6.5.2.9.3.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.5.2.9.3.6 DECOMPOSITION

None.

3.6.5.2.9.4 SEQUENTIALLY_UPDATE_COVARIANCE_MATRIX_AND_STATE_VECTOR (CATALOG #P135-0)

This LLCSC is a generic package which contains one procedure, "Update", which updates the Covariance Matrix, P, and state Vector, X.

3.6.5.2.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R152.

3.6.5.2.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table summarizes the generic formal data types required by this part:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_ Elements	floating point type	Elements making up the Kalman Filter aggregates
P_Matrices	private	Data type of P matrix
Measurement_ Variance_Vectors	vector	Vector indexed by Measurement Indices containing Kalman_Filter_ Elements
Measurement_Vectors	vector	Vector indexed by Measurement_ Indices containing Kalman_Filter_ Elements
P_Row_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
State_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
Compact_H_Matrices	vector	Data type of Compact H matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Element	function	Extracts an element of a P Matrix
Row	function	Extracts a row of a P matrix
"*"	function	A K Column Vector is multiplied by the transpose of a P Row vector, yielding a P matrix
"-"	function	Two P matrices are added, yielding a Symmetric matrix
"+"	function	Add a K column vector and a state vector, yielding a state vector
"*"	function	Multiply a K column vector by a Kalman Filter Element, yielding a K column vector

3.6.5.2.9.4.3 LOCAL ENTITIES

Packages:

The body of this package instantiates Part R149, Compute Kalman Gain, part R150, Update Error Covariance Matrix, and part R151, Update State Vector

3.6.5.2.9.4.4 INTERRUPTS

None.

3.6.5.2.9.4.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Compact_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package BDT       renames Basic_Data_Types;
   package KFCompact renames Kalman_Filter_Compact_H_Parts;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                     (State_Indices       => State_Indices,
                      Measurement_Indices => Measurement_Indices,
                      Intervals           => BDT.Seconds);
   use KDT;
   ...
   package SUCVASV is new KFCompact.
```

```
            Sequentially_Update_Covariance_Matrix_And_State_Vector
            (State_Indices                 => State_Indices,
             Measurement_Indices           => Measurement_Indices,
             Kalman_Filter_Elements        => Kalman_Filter_Elements,
             P_Matrices                    => KDT.N_by_N_Symmetric.Matrices,
             Measurement_Variance_Vectors  => KDT.M_by_1.Vectors,
             Measurement_Vectors           => KDT.M_by_1.Vectors,
             P_Row_Vectors                 => KDT.N_by_1.Vectors,
             K_Column_Vectors              => KDT.N_by_1.Vectors,
             State_Vectors                 => KDT.N_by_1.Vectors,
             Compact_H_Matrices            => KDT.M_by_1_Discrete_Vectors);
   ...
begin
   ...
   SUCVASV.Update (X                    => My_X,
                   P                    => My_P,
                   Z                    => My_Z,
                   Compact_H            => My_Compact_H,
                   Measurement_Variance => My_Measurement_Variance);
   ...
```

3.6.5.2.9.4.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.5.2.9.4.7 DECOMPOSITION

This LLCSC contains just the function "Update", which updates the Covariance Matrix and State Vector.

3.6.5.2.9.4.8 PART DESIGN

None.

3.6.5.2.9.5 KALMAN_UPDATE (CATALOG #P136-0)

This LLCSC is a generic package which contains 1 procedure, "Update" which updates the State Vector, X, given the old X vector, the Z vector, the K vector, the Measurement Number, and the Compact H array.

3.6.5.2.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R147.

3.6.5.2.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table summarizes the generic formal data types required by this part:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_Elements	floating point type	Elements making up the Kalman Filter aggregates
Phi_Matrices	private	Data type of Phi matrix
P_And_Q_Matrices	private	Data type of a P and Q matrix
Measurement_Variance_Vectors	vector	Vector indexed by Measurement_Indices containing Kalman_Filter_Elements
Measurement_Vectors	vector	Vector indexed by Measurement_Indices containing Kalman_Filter_Elements
P_Row_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
State_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
Compact_H_Matrices	vector	Data type of Compact H matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Element	function	Extracts an element of a P and Q Matrix
Row	function	Extracts a row of a P and Q matrix
ABA_Transpose	function	Performs an ABA transpose on a Phi Matrix and a P and Q Matrix
"*"	function	A K Column Vector is multiplied by the transpose of a P Row vector, yielding a P and Q matrix
"-"	function	Two P and Q matrices are added, yielding a P and Q matrix
"+"	function	Add a K column vector and a state vector, yielding a state vector
"*"	function	Multiply a K column vector by a Kalman Filter Element, yielding a K column vector

3.6.5.2.9.5.3 LOCAL ENTITIES

Packages:

The body of this package instantiates Part R152, Sequentially Update Covariance Matrix and State Vector, and Part R146, Error Covariance Matrix Manager

3.6.5.2.9.5.4 INTERRUPTS

None.

3.6.5.2.9.5.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Compact_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package BDT       renames Basic_Data_Types;
   package KFCompact renames Kalman_Filter_Compact_H_Parts;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                         (State_Indices       => State_Indices,
                          Measurement_Indices => Measurement_Indices,
```

```
                              Intervals          => BDT.Seconds);
   use KDT;
   ...
   package Kal_Update is new KFCompact.Kalman_Update
            (State_Indices                     => State_Indices,
             Measurement_Indices               => Measurement_Indices,
             Kalman_Filter_Elements            => Kalman_Filter_Elements,
             Phi_Matrices                      => KDT.N_by_N_Dynamically_Sparse_Matrice
             P_and_Q_Matrices                  => KDT.N_by_N_Symmetric.Matrices,
             Measurement_Variance_Vectors      => KDT.M_by_1.Vectors,
             Measurement_Vectors               => KDT.M_by_1.Vectors,
             P_Row_Vectors                     => KDT.N_by_1.Vectors,
             K_Column_Vectors                  => KDT.N_by_1.Vectors,
             State_Vectors                     => KDT.N_by_1.Vectors,
             Compact_H_Matrices                => KDT.M_by_1_Discrete_Vectors);
   ...
begin
   ...
   Kal_Update.Update (X                    => My_X,
                      P                    => My_P,
                      Z                    => My_Z,
                      Compact_H            => My_Compact_H,
                      Measurement_Variance => My_Measurement_Variance,
                      Propagated_Phi       => My_Phi,
                      Propagated_Q         => My_Q);
   ...
```

3.6.5.2.9.5.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.5.2.9.5.7 DECOMPOSITION

This LLCSC contains just the function "Update", which does the Kalman Update.

3.6.5.2.9.5.8 PART DESIGN

None.

3.6.5.2.9.6 UPDATE_ERROR_COVARIANCE_MATRIX_GENERAL_FORM

This unit is a generic procedure which computes the updated covariance matrix resulting from the processing of a single component of the measurement vector, Z.

3.6.5.2.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R150.

3.6.5.2.9.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table summarizes the generic formal data types required by this part:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_ Elements	floating point type	Elements contained in the Kalman Filter aggregates
P_Matrices	private	Type of covariance matrix. It represents a symmetric matrix index by (states, states)
K_H_Product_ Matrices	private	A matrix of the form I_KH, where K is a K_Column_Vector and H is a row of the H matrix. It represents a matrix indexed by (states, states)
Measurement_ Variance_	vector	Type of Measurement Variance Vector (R). It is indexed by (states)
K_Column_Vectors	vector	Type of a Column of the Kalman Gain Matrix (K). It is indexed by (states)
Compact_H_Matrices	vector	A vector, index by Measurement_ Indices containing the indices of the measured states. It represents a matrix indexed by (Measurement, states) that is all zeroes except for locations specified by row=I, column= Compact_H_Matrices(I)

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
I_Minus_Column_ Matrix	function	K and the current state measured (as indicated by whicvh element of the current row of H is a "1") => I_KH
ABA_Transpose	function	K_H_Product_Matrices * Kalman_Filter_ Elements * transpose(K_H_Product_ Matrices) => P_Matrices
ABA_Transpose	function	K_Column_Vectors * Kalman_Filter_ Elements * transpose(K_Column_ Vectors) => P_Matrices
"+"	function	P_Matrices + P_Matrices => P_Matrices

3.6.5.2.9.6.3 INTERRUPTS

None.

3.6.5.2.9.6.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Compact_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package BDT       renames Basic_Data_Types;
   package KFCompact renames Kalman_Filter_Compact_H_Parts;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                        (State_Indices       => State_Indices,
                         Measurement_Indices => Measurement_Indices,
                         Intervals           => BDT.Seconds);
   use KDT;
   ...
   procedure Update_P is new KFCompact.Update_Error_Covariance_Matrix
              (Measurement_Indices     => Measurement_Indices,
               State_Indices           => State_Indices,
               Kalman_Filter_Elements  => KDT.Kalman_Filter_Elements,
               P_Matrices              => KDT.N_by_N_Symmetric.Matrices,
               P_Row_Vectors           => KDT.N_By_1.Vectors,
               K_Column_Vectors        => KDT.N_By_1.Vectors,
               Compact_H_Matrices      => Compact_H_Vectors);
   ...
begin
   ...
   Update_P (P                    => My_P,
             Measurement_Number   => This_Measurement,
```

```
             K                     => My_K,
             Compact_H             => My_Compact_H);
   ...
```

3.6.5.2.9.6.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.5.2.9.6.6 DECOMPOSITION

None.

(This page left intentionally blank.)

```
package Kalman_Filter_Compact_H_Parts is

pragma PAGE;
   generic
      type State_Indices           is (<>);
      type Measurement_Indices     is (<>);
      type Kalman_Filter_Elements  is digits <>;

      type P_Matrices is private;

      type Measurement_Variance_Vectors is array (Measurement_Indices)
              of Kalman_Filter_Elements;
      type K_Column_Vectors is array (State_Indices)
              of Kalman_Filter_Elements;

      type Compact_H_Matrices is array (Measurement_Indices) of State_Indices;

      with function Element (Source : P_Matrices;
                             Row    : State_Indices;
                             Column : State_Indices)
                           return Kalman_Filter_Elements is <>;

   function Compute_Kalman_Gain
              (P                    : P_Matrices;
               Measurement_Number   : Measurement_Indices;
               Compact_H            : Compact_H_Matrices;
               Measurement_Variance : Measurement_Variance_Vectors)
            return K_Column_Vectors;

pragma PAGE;
   generic
      type State_Indices           is (<>);
      type Measurement_Indices     is (<>);
      type Kalman_Filter_Elements  is digits <>;
      type P_Matrices              is private;
      type P_Row_Vectors           is array (State_Indices)
                                      of Kalman_Filter_Elements;
      type K_Column_Vectors        is array (State_Indices)
                                      of Kalman_Filter_Elements;
      type Compact_H_Matrices      is array (Measurement_Indices)
                                      of State_Indices;
      with function Row (Source : P_Matrices;
                         Row    : State_Indices) return P_Row_Vectors is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : P_Row_Vectors) return P_Matrices is <>;
      with function "-" (Left  : P_Matrices;
                         Right : P_Matrices) return P_Matrices is <>;
   procedure Update_Error_Covariance_Matrix
               (P                  : in out P_Matrices;
                Measurement_Number : in     Measurement_Indices;
                K                  : in     K_Column_Vectors;
                Compact_H          : in     Compact_H_Matrices);

pragma PAGE;
   generic
      type State_Indices           is (<>);
      type Measurement_Indices     is (<>);
```

```
      type Kalman_Filter_Elements is digits <>;
      type Measurement_Vectors    is array (Measurement_Indices)
                                       of Kalman_Filter_Elements;
      type K_Column_Vectors       is array (State_Indices)
                                       of Kalman_Filter_Elements;
      type State_Vectors          is array (State_Indices)
                                       of Kalman_Filter_Elements;
      type Compact_H_Matrices     is array (Measurement_Indices)
                                       of State_Indices;
      with function "+" (Left  : K_Column_Vectors;
                         Right : State_Vectors) return State_Vectors is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : Kalman_Filter_Elements)
                        return K_Column_Vectors is <>;
   procedure Update_State_Vector
              (X                  : in out State_Vectors;
               Z                  : in     Measurement_Vectors;
               K                  : in     K_Column_Vectors;
               Measurement_Number : in     Measurement_Indices;
               Compact_H          : in     Compact_H_Matrices);

pragma PAGE;
   generic
      type State_Indices          is (<>);
      type Measurement_Indices    is (<>);
      type Kalman_Filter_Elements is digits <>;
      type P_Matrices is private;
      type Measurement_Variance_Vectors is array (Measurement_Indices)
                                             of Kalman_Filter_Elements;
      type Measurement_Vectors is array (Measurement_Indices)
                                    of Kalman_Filter_Elements;
      type P_Row_Vectors       is array (State_Indices)
                                    of Kalman_Filter_Elements;
      type K_Column_Vectors    is array (State_Indices)
                                    of Kalman_Filter_Elements;
      type State_Vectors       is array (State_Indices)
                                    of Kalman_Filter_Elements;
      type Compact_H_Matrices  is array (Measurement_Indices) of State_Indices;
      with function Element (Source : P_Matrices;
                             Row    : State_Indices;
                             Column : State_Indices)
                            return Kalman_Filter_Elements is <>;
      with function Row (Source : P_Matrices;
                         Row    : State_Indices) return P_Row_Vectors is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : P_Row_Vectors) return P_Matrices is <>;
      with function "-" (Left  : P_Matrices;
                         Right : P_Matrices) return P_Matrices is <>;
      with function "+" (Left  : K_Column_Vectors;
                         Right : State_Vectors)
                   return State_Vectors is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : Kalman_Filter_Elements)
                       return K_Column_Vectors is <>;

   package Sequentially_Update_Covariance_Matrix_And_State_Vector is
```

```
      procedure Update
                  (P                    : in out P_Matrices;
                   X                    : in out State_Vectors;
                   Z                    : in     Measurement_Vectors;
                   Compact_H            : in     Compact_H_Matrices;
                   Measurement_Variance : in     Measurement_Variance_Vectors);

   end Sequentially_Update_Covariance_Matrix_And_State_Vector;

pragma PAGE;
   generic
      type State_Indices           is (<>);
      type Measurement_Indices     is (<>);
      type Kalman_Filter_Elements is digits <>;

      type Phi_Matrices     is private;
      type P_And_Q_Matrices is private;

      type Measurement_Variance_Vectors is array (Measurement_Indices)
           of Kalman_Filter_Elements;
      type Measurement_Vectors is array (Measurement_Indices)
                                    of Kalman_Filter_Elements;
      type P_Row_Vectors       is array (State_Indices)
                                    of Kalman_Filter_Elements;
      type K_Column_Vectors    is array (State_Indices)
                                    of Kalman_Filter_Elements;
      type State_Vectors       is array (State_Indices)
                                    of Kalman_Filter_Elements;
      type Compact_H_Matrices  is array (Measurement_Indices) of State_Indices;

      with function Element (Source : P_And_Q_Matrices;
                             Row    : State_Indices;
                             Column : State_Indices)
                            return Kalman_Filter_Elements is <>;
      with function Row (Source : P_And_Q_Matrices;
                         Row    : State_Indices) return P_Row_Vectors is <>;
      with function Aba_Transpose (Phi : Phi_Matrices;
                                   P   : P_And_Q_Matrices)
                                  return P_And_Q_Matrices is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : P_Row_Vectors) return P_And_Q_Matrices is <>;
      with function "*" (Left  : Phi_Matrices;
                         Right : State_Vectors) return State_Vectors is <>;
      with function "-" (Left  : P_And_Q_Matrices;
                         Right : P_And_Q_Matrices)
                        return P_And_Q_Matrices is <>;
      with function "+" (Left  : K_Column_Vectors;
                         Right : State_Vectors)
                      return State_Vectors is <>;
      with function "+" (Left  : P_And_Q_Matrices;
                         Right : P_And_Q_Matrices)
                      return P_And_Q_Matrices is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : Kalman_Filter_Elements)
                      return K_Column_Vectors is <>;

   package Kalman_Update is
```

```
      procedure Update
                (X                      : in out State_Vectors;
                 P                      : in out P_And_Q_Matrices;
                 Z                      : in     Measurement_Vectors;
                 Compact_H              : in     Compact_H_Matrices;
                 Measurement_Variance   : in     Measurement_Variance_Vectors;
                 Propagated_Phi         : in     Phi_Matrices;
                 Propagated_Q           : in     P_And_Q_Matrices);

   end Kalman_Update;

pragma PAGE;
   generic
      type State_Indices           is (<>);
      type Measurement_Indices     is (<>);
      type Kalman_Filter_Elements  is digits <>;

      type P_Matrices           is private;
      type K_H_Product_Matrices is private;

      type Measurement_Variance_Vectors is array (Measurement_Indices)
                                       of Kalman_Filter_Elements;

      type K_Column_Vectors   is array (State_Indices)
                                  of Kalman_Filter_Elements;
      type Compact_H_Matrices is array (Measurement_Indices) of State_Indices;
      with function I_Minus_Column_Matrix (K             : K_Column_Vectors;
                                           State_Measure : State_Indices)
                                          return K_H_Product_Matrices is <>;
      with function Aba_Transpose (A : K_H_Product_Matrices;
                                   B : P_Matrices)
                                  return P_Matrices is <>;
      with function Aba_Transpose (A : K_Column_Vectors;
                                   B : Kalman_Filter_Elements)
                                  return P_Matrices is <>;
      with function "+" (Left  : P_Matrices;
                         Right : P_Matrices) return P_Matrices is <>;

   procedure Update_Error_Covariance_Matrix_General_Form
             (P                    : in out P_Matrices;
              Measurement_Number   : in     Measurement_Indices;
              K                    : in     K_Column_Vectors;
              Compact_H            : in     Compact_H_Matrices;
              Measurement_Variance : in     Measurement_Variance_Vectors);

end Kalman_Filter_Compact_H_Parts;
```

3.6.5.3 KALMAN_FILTER_COMPLICATED_H_PARTS TLCSC (CATALOG #P143-0)

This part, which is designed as an Ada package, contains specifications for all CAMP parts which can be used to implement a Kalman Filter when a complicated Measurement Sensitivity Matrix (Complicated H Matrix) is used.

3.6.5.3.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this TLCSC:

Name	Requirements Allocation
Compute_Kalman_Gain	R182
Update_Error_Covariance_Matrix	R183
Update_State_Vector	R184
Sequentially_Update_Covariance_Matrix_and_ State_Vector	R201
Kalman_Update	R181
Update_Error_Covariance_Matrix_General_Form	

3.6.5.3.2 INPUT/OUTPUT

None.

3.6.5.3.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.5.3.4 LOCAL ENTITIES

None.

3.6.5.3.5 INTERRUPTS

None.

3.6.5.3.6 TIMING AND SEQUENCING

None.

3.6.5.3.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.5.3.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Compute_Kalman_Gain	generic function	Computes the Kalman gain vector resulting from the processing of a single component of measurement vector,Z
Update_Error_Covariance_Matrix	generic procedure	Computes the Error Covariance Matrix resulting from the processing of a single component of measurement vector,Z
Update_State_Vector	generic procedure	Computes the State Vector resulting from the processing of a single component of measurement vector,Z
Sequentially_Update_Error_Covariance_Matrix_And_State_Vector	generic package	Computes the updated Covariance Matrix, P, and state Vector, X.
Kalman_Update	generic package	Compute the updated State Vector, X, given the old X vector, the Z vector, the K vector, the Measurement Number, and the Compact H array.
Update_Error_Covariance_Matrix_General_Form	generic procedure	Computes the Error Covariance Matrix resulting from the processing of a single component of measurement vector,Z using the general form

3.6.5.3.9 PART DESIGN

3.6.5.3.9.1 COMPUTE_KALMAN_GAIN (CATALOG #P144-0)

This LLCSC is a generic function which computes the Kalman gain vector resulting from the processing of a single component of the measurement vector, Z.

3.6.5.3.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R182.

3.6.5.3.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_ Elements	floating point type	Elements contained in the Kalman Filter aggregates
P_Matrices	private	Data type for n x n P matrixx
H_Matrices	private	Data type for n x n H matrix
H_Row_Vectors	private	Data type of vector representing an H vector
Measurement_ Variance_ Vectors	array	Vector indexed by Measurement_ Indices containing Kalman_Filter_ Elements
K_Column_Vectors	array	Vector indexed by State_Indices containing Kalman_Filter_Elements

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Active_H_Vector	function	Returns the row of an (MxN) H matrix that is referenced
Times_Transpose	function	Multiplies a P matrix by the transpose of a n x 1 H Row Vector, yielding a K Column Vector
Dot_Product	function	Multiplies a (Nx1) H Row Vector by a K Column Vector, yielding a Kalman Filter Element
"/"	function	Divides a (Nx1) K Column Vector by a Kalman Filter Element, yielding a K Column Vector

3.6.5.3.9.1.3 INTERRUPTS

None.

3.6.5.3.9.1.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Complicated_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package KFCompx renames Kalman_Filter_Complicated_H_Parts;
   package BDT     renames Basic_Data_Types;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                      (State_Indices       => State_Indices,
                       Measurement_Indices => Measurement_Indices,
                       Intervals           => BDT.Seconds);
   use KDT;
   ...
   function CKG is new KFCompx.Compute_Kalman_Gain
              (State_Indices           => State_Indices,
               Measurement_Indices     => Measurement_Indices,
               Kalman_Filter_Elements  => KDT.Kalman_Filter_Elements,
               P_Matrices              => KDT.N_by_N_Symmetric.Matrices,
               H_Matrices              => KDT.M_By_N_Statically_Sparse.Matrices;
               H_Row_Vectors           => KDT.N_By_1.Vectors,
               Measurement_Variance_Vectors
                                       => KDT.M_By_1.Vectors,
               K_Column_Vectors        => KDT.N_By_1.Vectors);
   ...
begin
   ...
   My_K := CKG (P                    => My_P,
                Measurement_Number   => This_Measurement,
                Complicated_H        => My_Complicated_H,
                Measurement_Variance => My_Measurement_Variance);
   ...
```

3.6.5.3.9.1.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.5.3.9.1.6 DECOMPOSITION

None.

3.6.5.3.9.2 UPDATE_ERROR_COVARIANCE_MATRIX (CATALOG #P145-0)

This unit is a generic procedure which computes the updated covariance matrix resulting from the processing of a single component of the measurement vector, Z.

3.6.5.3.9.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R183.

3.6.5.3.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_Elements	floating point type	Elements contained in the Kalman Filter aggregates
P_Matrices	private	Data type for private n x n P matrix
H_Matrices	private	Data type for private H matrix
H_Row_Vectors	private	Private vector representing a row of the H matrix
K_H_Product_Matrices	array	Private matrix representing product of K and H matrices
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Active_H_Vector	function	Returns the row of a (MxN) H Matrix that is referenced
Subtract_From_ Identity	procedure	Subtracts a K_H_Product matrix from the Identity Matrix (i.e., I - S)
"*"	function	Multiplies a K Column Vector by a H Row Vector, yielding a K H Product Matrix
"*"	function	Multiplies a K H Product Matrix by a P Matrix, yielding a P Matrix

3.6.5.3.9.2.3 INTERRUPTS

None.

3.6.5.3.9.2.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Complicated_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package KFCompx renames Kalman_Filter_Complicated_H_Parts;
   package BDT     renames Basic_Data_Types;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                        (State_Indices       => State_Indices,
                         Measurement_Indices => Measurement_Indices,
                         Intervals           => BDT.Seconds);
   use KDT;
   ...
   procedure Update_P is new KFCompx.Update_Error_Covariance_Matrix
               (State_Indices           => State_Indices,
                Measurement_Indices     => Measurement_Indices,
                Kalman_Filter_Elements  => KDT.Kalman_Filter_Elements,
                P_Matrices              => KDT.N_by_N_Symmetric.Matrices,
                H_Matrices              => KDT.M_By_N_Statically_Sparse.Matrices,
                H_Row_Vectors           => KDT.N_By_1.Vectors,
                K_H_Product_Matrices    => KDT.N_by_N_Dynamically_Sparse_Matrices,
                K_Column_Vectors        => KDT.N_By_1.Vectors);
   ...
begin
   ...
   Update_P (P                   => My_P,
             Complicated_H       => My_Complicated_H,
```

```
            Measurement_Number   => This_Measurement,
            K                    => My_K);
   ...
```

3.6.5.3.9.2.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.5.3.9.2.6 DECOMPOSITION

None.

3.6.5.3.9.3 UPDATE_STATE_VECTOR (CATALOG #P146-0)

This unit is a generic procedure which updates the State Vector, X, given the old X vector, the Z vector, the K vector, the Measurement Number, and the Complicated H array.

3.6.5.3.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R184.

3.6.5.3.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_Elements	floating point type	Elements contained in the Kalman Filter aggregates
H_Matrices	private	Data type of m x n H matrix
H_Row_Vectors	private	Vector representing a row of an H matrix
Measurement_Vectors	vector	Vector indexed by Measurement Indices
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
State_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Active_H_Vector	function	Returns the row of a (MxN) H matrix that is referenced
"*"	function	Computes the product of an H Row vector and a State vector, yeilding a Kalman Filter Element
Dot_Product	function	Multiplies a (Nx1) K Column Vector by a Kalman Filter Element, yielding a K Column Vector
"+"	function	Adds a state vector and a K column vector (both n x 1) yielding a state vector

3.6.5.3.9.3.3 INTERRUPTS

None.

3.6.5.3.9.3.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Complicated_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package KFCompx renames Kalman_Filter_Complicated_H_Parts;
   package BDT     renames Basic_Data_Types;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                          (State_Indices       => State_Indices,
                           Measurement_Indices => Measurement_Indices,
                           Intervals           => BDT.Seconds);
   use KDT;
   ...
   package USV is new KFCompx.Update_State_Vector
                 (State_Indices           => State_Indices,
                  Measurement_Indices     => Measurement_Indices,
                  Kalman_Filter_Elements  => KDT.Kalman_Filter_Elements,
                  H_Matrices              => KDT.M_By_N_Statically_Sparse.Matrices,
                  H_Row_Vectors           => KDT.N_By_1.Vectors,
                  Measurement_Vectors     => KDT.M_By_1.Vectors,
                  K_Column_Vectors        => KDT.N_By_1.Vectors,
                  State_Vectors           => KDT.N_By_1.Vectors);
   ...
begin
   ...
   USV (X                      => My_X,
        Z                      => My_Z,
        Complicated_H          => My_Complicated_H,
        Measurement_Number     => This_Measurement,
        Complicated_H          => My_H);
   ...
```

3.6.5.3.9.3.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.5.3.9.3.6 DECOMPOSITION

None.

3.6.5.3.9.4 SEQUENTIALLY UPDATE COVARIANCE MATRIX AND STATE VECTOR (CATALOG #P147-0)

This LLCSC is a generic package which contains 1 procedure, "Update", which updates the Covariance Matrix, P, and state Vector, X.

3.6.5.3.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R152.

3.6.5.3.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Base Type	Description
States_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_ Elements	floating point type	Elements contained in the Kalman Filter aggregates
P_Matrices	private	Data type for n x n private P matrix
H_Matrices	private	Data type for private m x n H matrix
K_H_Product_ Matrices	private	Data type for private n x n K and H matrix
H_Row_Vectors	private	Vector representing a row of a P matrix
Measurement_Vectors	vector	Vector indexed by Measurement_ Indices containing Kalman_Filter_ Elements
Measurement_ Variance_ Vectors	vector	Vector indexed by Measurement_ Indices containing Kalman_Filter_ Elements
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
State_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Active_H_Vector	function	Returns the row of an (MxN) H matrix that is referenced
Subtract_From_Identity	procedure	Subtracts a K and H Product matrix from the Identity Matrix(i.e., I - S)
Times_Transpose	function	Multiplies a P matrix by the transpose of a H Row Vector, yielding a K Column Vector
Dot_Product	function	Multiplies a H Row Vector by a K Column vector, yielding a Kalman Filter Element
Dot_Product	function	Multiplies a H Row Vector by a State vector, yielding a Kalman Filter Element
"*"	function	Multiplies a K Column Vector by a H Row Vector, yielding a K and H Product Matrix
"*"	function	Multiplies a K and H Product Matrix by a P Matrix, yielding a P Matrix
"*"	function	Multiplies a K Column Vector by a Kalman Filter Element, yielding a K Column Vector
"/"	function	Divides a K Column Vector by a Kalman Filter Element, yielding a K Column Vector
"+"	function	Adds a State Vector and a K Column Vector, yielding a State Vector

3.6.5.3.9.4.3 LOCAL ENTITIES

None.

3.6.5.3.9.4.4 INTERRUPTS

None.

3.6.5.3.9.4.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Complicated_H_Parts;
with Kalman_Filter_Data_Types;
```

```
with Basic_Data_Types;
...
   package KFCompx renames Kalman_Filter_Complicated_H_Parts;
   package BDT     renames Basic_Data_Types;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                      (State_Indices       => State_Indices,
                       Measurement_Indices => Measurement_Indices,
                       Intervals           => BDT.Seconds);
   use KDT;

  package SUCVASV is new
        KFCompx.Sequentially_Update_Covariance_Matrix_And_State_Vector
           (State_Indices           => State_Indices,
            Measurement_Indices     => Measurement_Indices,
            Kalman_Filter_Elements => KDT.Kalman_Filter_Elements,
            P_Matrices              => KDT.N_by_N_Symmetric.Matrices,
            H_Matrices              => KDT.M_By_N_Statically_Sparse.Matrices,
            K_H_Product_Matrices    => KDT.N_by_N_Dynamically_Sparse_Matrices,
            H_Row_Vectors           => KDT.N_By_1.Vectors,
            Measurement_Variance_Vectors
                                    => KDT.M_By_1.Vectors,
            Measurement_Vectors     => KDT.M_By_1.Vectors,
            K_Column_Vectors        => KDT.N_By_1.Vectors,
            State_Vectors           => KDT.N_By_1.Vectors);
   ...
begin .
   ...
   SUCVASV.Update (X                    => My_X,
                   P                    => My_P,
                   Z                    => My_Z,
                   Complicated_H        => My_Complicated_H,
                   Measurement_Variance => My_Measurement_Variance);
   ...
```

3.6.5.3.9.4.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.5.3.9.4.7 DECOMPOSITION

None.

3.6.5.3.9.4.8 PART DESIGN

None.

3.6.5.3.9.5 KALMAN_UPDATE (CATALOG #P148-0)

This LLCSC is a generic package which updates the State Vector, X, given the old X vector, the Z vector, the K vector, the Measurement Number, and the Complicated H array.

3.6.5.3.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R181.

Context of tlcsc: This part with's no library units.

3.6.5.3.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Base Type	Description
States_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_ Elements	floating point type	Elements contained in the Kalman Filter aggregates
Phi_Matrices	private	Data type for private n x n Phi matrix
P_Matrices	private	Data type for n x n private P matrix
H_Matrices	private	Data type for private m x n H matrix
K_H_Product_ Matrices	private	Data type for private n x n K and H matrix
H_Row_Vectors	private	Vector representing a row of a P matrix
Measurement_Vectors	vector	Vector indexed by Measurement_ Indices containing Kalman_Filter_ Elements
Measurement_ Variance_ Vectors	vector	Vector indexed by Measurement_ Indices containing Kalman_Filter_ Elements
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements
State_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Active_H_Vector	function	Returns the row of an (MxN) H matrix that is referenced
Subtract_From_Identity	procedure	Subtracts a K and H Product matrix from the Identity Matrix(i.e., I - S)
Times_Transpose	function	Multiplies a P matrix by the transpose of a H Row Vector, yielding a K Column Vector
ABA_Transpose	function	Does an ABA Transpose opertion on a Phi matrix and a P and Q matrix, yielding a P and Q matrix
Dot_Product	function	Multiplies a H Row Vector by a K Column vector, yielding a Kalman Filter Element
Dot_Product	function	Multiplies a H Row Vector by a State vector, yielding a Kalman Filter Element
"*"	function	Multiplies a K Column Vector by a H Row Vector, yielding a K and H Product Matrix
"*"	function	Multiplies a K and H Product Matrix by a P Matrix, yielding a P Matrix
"*"	function	Multiplies a K Column Vector by a Kalman Filter Element, yielding a K Column Vector
"*"	function	Multiplies a Phi Matrix by a State Vector yielding a State Vector
"/"	function	Divides a K Column Vector by a Kalman Filter Element, yielding a K Column Vector
"+"	function	Adds a State Vector and a K Column Vector, yielding a State Vector

3.6.5.3.9.5.3 LOCAL ENTITIES

Packages:

The body of this package instantiates Part R201, Sequentially Update Covariance Matrix and State Vector, and Part R146, Error Covariance Matrix Manager

3.6.5.3.9.5.4 INTERRUPTS

None.

3.6.5.3.9.5.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Complicated_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package KFCompx renames Kalman_Filter_Complicated_H_Parts;
   package BDT     renames Basic_Data_Types;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                       (State_Indices       => State_Indices,
                        Measurement_Indices => Measurement_Indices,
                        Intervals           => BDT.Seconds);
   use KDT;
   ...
   package SUCVASV is new KFCompx.Kalman_Update
              (State_Indices           => State_Indices,
               Measurement_Indices     => Measurement_Indices,
               Kalman_Filter_Elements  => KDT.Kalman_Filter_Elements,
               Phi_Matrices            => KDT.N_by_N_Dynamically_Sparse_Matrices,
               P_and_Q_Matrices        => KDT.N_by_N_Symmetric.Matrices,
               H_Matrices              => KDT.M_By_N_Statically_Sparse.Matrices,
               K_H_Product_Matrices    => KDT.N_by_N_Dynamically_Sparse_Matrices,
               H_Row_Vectors           => KDT.N_By_1.Vectors,
               Measurement_Variance_Vectors
                                       => KDT.M_By_1.Vectors,
               Measurement_Vectors     => KDT.M_By_1.Vectors,
               K_Column_Vectors        => KDT.N_By_1.Vectors,
               State_Vectors           => KDT.N_By_1.Vectors);
   ...
begin
   ...
   SUCVASV.Update (X                    => My_X,
                   P                    => My_P,
                   Z                    => My_Z,
                   Complicated_H        => My_Complicated_H,
                   Measurement_Variance => My_Measurement_Variance,
                   Propagated_Phi       => Phi_Matrices,
                   Propagated_Q         => P_And_Q_Matrices);
   ...
```

3.6.5.3.9.5.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.5.3.9.5.7 DECOMPOSITION

This LLCSC contains just the function "Update", which does the Kalman Update.

3.6.5.3.9.5.8 PART DESIGN

None.

3.6.5.3.9.6 UPDATE_ERROR_COVARIANCE_MATRIX_GENERAL_FORM

This unit is a generic procedure which computes the updated covariance matrix resulting from the processing of a single component of the measurement vector, Z. This routine uses the general form of the calculation.

3.6.5.3.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R183.

3.6.5.3.9.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Base Type	Description
State_Indices	discrete	Index to the arrays which depend on the number of states
Measurement_Indices	discrete	Index to the arrays which depend on the number of measurements
Kalman_Filter_Elements	floating point type	Elements contained in the Kalman Filter aggregates
P_Matrices	private	Data type for private n x n P matrix
H_Matrices	private	Data type for private H matrix
H_Row_Vectors	private	Private vector representing a row of the H matrix
K_H_Product_Matrices	array	Private matrix representing product of K and H matrices
Measurement_Variance_Vectors	array	Vector indexed by Measurement Indices containing Kalman Filter Elements
K_Column_Vectors	vector	Vector indexed by State_Indices containing Kalman_Filter_Elements

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Active_H_Vector	function	Returns the row of a (MxN) H Matrix that is referenced
Subtract_From_ Identity	procedure	Subtracts a K_H_Product matrix from the Identity Matrix (i.e., I - S)
ABA_Transpose	function	Does an ABA transpose on a K H Matrix and a P Matrix, yielding a P Matrix
ABA_Transpose	function	Does an ABA transpose on a K Column Vector and a Kalman Filter Element, yielding a P Matrix
"*"	function	Multiplies a K Column Vector by a H Row Vector, yielding a K H Product Matrix
"+"	function	Adds two P Matrices, yielding a P Matrix

3.6.5.3.9.6.3 INTERRUPTS

None.

3.6.5.3.9.6.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Kalman_Filter_Complicated_H_Parts;
with Kalman_Filter_Data_Types;
with Basic_Data_Types;
...
   package KFCompx renames Kalman_Filter_Complicated_H_Parts;
   package BDT     renames Basic_Data_Types;
   ...
   type State_Indices       is range 1 .. 27;
   type Measurement_Indices is range 1 .. 5;
   ...
   package KDT is new Kalman_Filter_Data_Types
                        (State_Indices       => State_Indices,
                         Measurement_Indices => Measurement_Indices,
                         Intervals           => BDT.Seconds);
   use KDT;
   ...
   procedure Update_P is new KFCompx.Update_Error_Covariance_Matrix
             (State_Indices           => State_Indices,
              Measurement_Indices     => Measurement_Indices,
              Kalman_Filter_Elements => KDT.Kalman_Filter_Elements,
              P_Matrices              => KDT.N_by_N_Symmetric.Matrices,
              H_Matrices              => KDT.M_By_N_Statically_Sparse.Matrices,
```

```
               H_Row_Vectors             => KDT.N_By_1.Vectors,
               K_H_Product_Matrices      => KDT.N_by_N_Dynamically_Sparse_Matrices,
               Measurement_Variance_Vectors
                                         => KDT.M_By_1.Vectors;
               K_Column_Vectors          => KDT.N_By_1.Vectors);
   ...
begin
   ...
   Update_P (P                      => My_P,
             Complicated_H          => My_Complicated_H,
             Measurement_Number     => This_Measurement,
             K                      => My_K);
   ...
```

3.6.5.3.9.6.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.5.3.9.6.6 DECOMPOSITION

None.

```
package Kalman_Filter_Complicated_H_Parts is

pragma PAGE;
   generic
      type State_Indices                  is (<>);
      type Measurement_Indices            is (<>);
      type Kalman_Filter_Elements         is digits <>;
      type P_Matrices                     is private;
      type H_Matrices                     is private;
      type H_Row_Vectors                  is private;
      type Measurement_Variance_Vectors is array (Measurement_Indices)
                                             of Kalman_Filter_Elements;
      type K_Column_Vectors               is array (State_Indices)
                                             of Kalman_Filter_Elements;
      with function Active_H_Vector (Source : H_Matrices;
                                     Row    : Measurement_Indices)
                                    return H_Row_Vectors is <>;
      with function Times_Transpose (Left  : P_Matrices;
                                     Right : H_Row_Vectors)
                                    return K_Column_Vectors is <>;
      with function Dot_Product (Left  : H_Row_Vectors;
                                 Right : K_Column_Vectors)
                                return Kalman_Filter_Elements is <>;
      with function "/" (Left  : K_Column_Vectors;
                         Right : Kalman_Filter_Elements)
                        return K_Column_Vectors is <>;
   function Compute_Kalman_Gain
             (P                    : P_Matrices;
              Measurement_Number   : Measurement_Indices;
              Complicated_H        : H_Matrices;
              Measurement_Variance : Measurement_Variance_Vectors)
             return K_Column_Vectors;

pragma PAGE;
   generic
      type State_Indices          is (<>);
      type Measurement_Indices    is (<>);
      type Kalman_Filter_Elements is digits <>;
      type P_Matrices             is private;
      type H_Matrices             is private;
      type H_Row_Vectors          is private;
      type K_H_Product_Matrices   is private;
      type K_Column_Vectors       is array (State_Indices)
                                     of Kalman_Filter_Elements;
      with function Active_H_Vector (Source : H_Matrices;
                                     Row    : Measurement_Indices)
                                    return H_Row_Vectors is <>;
      with function Subtract_From_Identity (Right : K_H_Product_Matrices)
                                           return K_H_Product_Matrices is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : H_Row_Vectors)
                        return K_H_Product_Matrices is <>;
      with function "*" (Left  : K_H_Product_Matrices;
                         Right : P_Matrices)
                        return P_Matrices is <>;
   procedure Update_Error_Covariance_Matrix
             (P                    : in out P_Matrices;
```

```
                 Measurement_Number : in     Measurement_Indices;
                 K                  : in     K_Column_Vectors;
                 Complicated_H      : in     H_Matrices);

pragma PAGE;
   generic
      type State_Indices          is (<>);
      type Measurement_Indices    is (<>);
      type Kalman_Filter_Elements is digits <>;
      type H_Matrices             is private;
      type H_Row_Vectors          is private;
      type Measurement_Vectors    is array (Measurement_Indices)
                                       of Kalman_Filter_Elements;
      type K_Column_Vectors       is array (State_Indices)
                                       of Kalman_Filter_Elements;
      type State_Vectors          is array (State_Indices)
                                       of Kalman_Filter_Elements;
      with function Active_H_Vector (Source : H_Matrices;
                                     Row    : Measurement_Indices)
                                    return H_Row_Vectors is <>;
      with function Dot_Product (Left  : H_Row_Vectors;
                                 Right : State_Vectors)
                                return Kalman_Filter_Elements is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : Kalman_Filter_Elements)
                        return K_Column_Vectors is <>;
      with function "+" (Left  : K_Column_Vectors;
                         Right : State_Vectors)
                        return State_Vectors is <>;
   procedure Update_State_Vector
                (X                  : in out State_Vectors;
                 Z                  : in     Measurement_Vectors;
                 K                  : in     K_Column_Vectors;
                 Measurement_Number : in     Measurement_Indices;
                 Complicated_H      : in     H_Matrices);

pragma PAGE;
   generic
      type State_Indices                is (<>);
      type Measurement_Indices          is (<>);
      type Kalman_Filter_Elements       is digits <>;
      type P_Matrices                   is private;
      type H_Matrices                   is private;
      type K_H_Product_Matrices         is private;
      type H_Row_Vectors                is private;
      type Measurement_Variance_Vectors is array (Measurement_Indices)
                                             of Kalman_Filter_Elements;
      type Measurement_Vectors          is array (Measurement_Indices)
                                             of Kalman_Filter_Elements;
      type K_Column_Vectors is array (State_Indices) of Kalman_Filter_Elements;
      type State_Vectors    is array (State_Indices) of Kalman_Filter_Elements;
      with function Active_H_Vector (Source : H_Matrices;
                                     Row    : Measurement_Indices)
                                    return H_Row_Vectors is <>;
      with function Subtract_From_Identity (Right : K_H_Product_Matrices)
                                    return K_H_Product_Matrices is <>;
      with function Times_Transpose (Left  : P_Matrices;
```

```
                                         Right : H_Row_Vectors)
                                        return K_Column_Vectors is <>;
      with function Dot_Product (Left  : H_Row_Vectors;
                                 Right : K_Column_Vectors)
                                return Kalman_Filter_Elements is <>;
      with function Dot_Product (Left  : H_Row_Vectors;
                                 Right : State_Vectors)
                                return Kalman_Filter_Elements is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : H_Row_Vectors)
                        return K_H_Product_Matrices is <>;
      with function "*" (Left  : K_H_Product_Matrices;
                         Right : P_Matrices)
                        return P_Matrices is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : Kalman_Filter_Elements)
                        return K_Column_Vectors is <>;
      with function "/" (Left  : K_Column_Vectors;
                         Right : Kalman_Filter_Elements)
                        return K_Column_Vectors is <>;
      with function "+" (Left  : K_Column_Vectors;
                         Right : State_Vectors)
                        return State_Vectors is <>;
   package Sequentially_Update_Covariance_Matrix_And_State_Vector is

      procedure Update
                   (P                      : in out P_Matrices;
                    X                      : in out State_Vectors;
                    Z                      : in     Measurement_Vectors;
                    Complicated_H          : in     H_Matrices;
                    Measurement_Variance   : in     Measurement_Variance_Vectors);

   end Sequentially_Update_Covariance_Matrix_And_State_Vector;

pragma PAGE;
   generic
      type State_Indices          is (<>);
      type Measurement_Indices    is (<>);
      type Kalman_Filter_Elements is digits <>;
      type Phi_Matrices           is private;
      type P_And_Q_Matrices       is private;
      type H_Matrices             is private;
      type K_H_Product_Matrices   is private;
      type H_Row_Vectors          is private;
      type Measurement_Variance_Vectors is array (Measurement_Indices)
                                            of Kalman_Filter_Elements;
      type Measurement_Vectors is array (Measurement_Indices)
                                            of Kalman_Filter_Elements;
      type K_Column_Vectors is array (State_Indices) of Kalman_Filter_Elements;
      type State_Vectors    is array (State_Indices) of Kalman_Filter_Elements;
      with function Active_H_Vector (Source : H_Matrices;
                                     Row    : Measurement_Indices)
                                    return H_Row_Vectors is <>;
      with function Subtract_From_Identity (Right : K_H_Product_Matrices)
                                           return K_H_Product_Matrices is <>;
      with function Times_Transpose (Left  : P_And_Q_Matrices;
                                     Right : H_Row_Vectors)
```

```
                                 return K_Column_Vectors is <>;
      with function Aba_Transpose (Phi : Phi_Matrices;
                                   P   : P_And_Q_Matrices)
                                 return P_And_Q_Matrices is <>;
      with function Dot_Product (Left  : H_Row_Vectors;
                                 Right : K_Column_Vectors)
                                return Kalman_Filter_Elements is <>;
      with function Dot_Product (Left  : H_Row_Vectors;
                                 Right : State_Vectors)
                                return Kalman_Filter_Elements is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : H_Row_Vectors)
                        return K_H_Product_Matrices is <>;
      with function "*" (Left  : K_H_Product_Matrices;
                         Right : P_And_Q_Matrices)
                        return P_And_Q_Matrices is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : Kalman_Filter_Elements)
                        return K_Column_Vectors is <>;
      with function "*" (Left  : Phi_Matrices;
                         Right : State_Vectors)
                        return State_Vectors is <>;
      with function "/" (Left  : K_Column_Vectors;
                         Right : Kalman_Filter_Elements)
                        return K_Column_Vectors is <>;
      with function "+" (Left  : K_Column_Vectors;
                         Right : State_Vectors)
                        return State_Vectors is <>;
      with function "+" (Left  : P_And_Q_Matrices;
                         Right : P_And_Q_Matrices)
                        return P_And_Q_Matrices is <>;

   package Kalman_Update is

      procedure Update (X                    : in out State_Vectors;
                        P                    : in out P_And_Q_Matrices;
                        Z                    : in     Measurement_Vectors;
                        Complicated_H        : in     H_Matrices;
                        Measurement_Variance : in     Measurement_Variance_Vectors;
                        Propagated_Phi       : in     Phi_Matrices;
                        Propagated_Q         : in     P_And_Q_Matrices);

   end Kalman_Update;

pragma PAGE;
   generic
      type State_Indices          is (<>);
      type Measurement_Indices    is (<>);
      type Kalman_Filter_Elements is digits <>;
      type P_Matrices             is private;
      type H_Matrices             is private;
      type H_Row_Vectors          is private;
      type K_H_Product_Matrices   is private;

      type Measurement_Variance_Vectors is array( Measurement_Indices )
           of Kalman_Filter_Elements;
```

```
      type K_Column_Vectors is array (State_Indices)
              of Kalman_Filter_Elements;

      with function Active_H_Vector (Source : H_Matrices;
                                     Row    : Measurement_Indices)
                                    return H_Row_Vectors is <>;
      with function Subtract_From_Identity (Right : K_H_Product_Matrices)
                                           return K_H_Product_Matrices is <>;
      with function Aba_Transpose (A : K_H_Product_Matrices;
                                   B : P_Matrices ) return P_Matrices is <>;
      with function Aba_Transpose (A : K_Column_Vectors;
                                   B : Kalman_Filter_Elements )
                                  return P_Matrices is <>;
      with function "*" (Left  : K_Column_Vectors;
                         Right : H_Row_Vectors)
                        return K_H_Product_Matrices is <>;
      with function "+" (Left  : P_Matrices;
                         Right : P_Matrices)
                        return P_Matrices is <>;

   procedure Update_Error_Covariance_Matrix_General_Form
             ( P                    : in out P_Matrices;
               Measurement_Number   : in     Measurement_Indices;
               K                    : in     K_Column_Vectors;
               Complicated_H        : in     H_Matrices;
               Measurement_Variance : in     Measurement_Variance_Vectors );

end Kalman_Filter_Complicated_H_Parts;
```

(This page left intentionally blank.)

3.6.6 GUIDANCE AND CONTROL

(This page left intentionally blank.)

3.6.6.1 WAYPOINT_STEERING TLCSC (CATALOG #P99-0)

This package contains the CAMP parts required to do the waypoint steering portion of navigation.

The following three waypoints are required to perform waypoint steering:

- o A : the last waypoint passed by the missile
- o B : the waypoint to which the missile is currently heading
- o C : the next waypoint to which the missile will head

3.6.6.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Steering_Vector_Operations	R170, R171
Steering_Vector_Operations_with_Arcsin	
Compute_Turn_Angle_and_Direction	R172
Crosstrack_And_Heading_Error_Operations	R173, R174, R175
Distance_to_Current_Waypoint	R176
Distance_to_Current_Waypoint_with_Arcsin	
Compute_Turning_and_Nonturning_Distances	R177
Turn_Test_Operations	R178, R179, R180

3.6.6.1.2 INPUT/OUTPUT

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following table describes the data types exported by this part:

Name	Range	Description
Turning_Directions	Left_Turn, Right_Turn	Indicates if the missile needs to make a right or a left-hand turn to go to the next waypoint
Turning_Statuses	Turning, Not_Turning	Indicates whether or not the missile is currently turning

3.6.6.1.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.6.1.4 LOCAL ENTITIES

None.

3.6.6.1.5 INTERRUPTS

None.

3.6.6.1.6 TIMING AND SEQUENCING

None.

3.6.6.1.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.6.1.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Steering_Vector_ Operations	generic package	Provides the capability to initialize and update the steering vectors; uses the assumption that alpha= sin(alpha) when computing segment BC distance
Steering_Vector_ Operations_with_ Arcsin	generic package	Provides the capability to initialize and update the steering vectors; does not use the assumption that alpha=sin(alpha) when computing segment BC distance
Compute_Turn_Angle_ and_Direction	generic procedure	Computes the tangent of 1/2 the turn angle, along with the turn direction
Crosstrack_and_ Heading_Error_ Operations	generic package	Computes the crosstrack and heading error for a missile in turning and nonturning flight
Distance_to_Current_ Waypoint	generic function	Computes the distance from missile position to current waypoint, B; uses the assumption that alpha= sin(alpha) when doing its computations
Distance_to_Current_ Waypoint_with_ Arcsin	generic function	Computes the distance from missile position to current waypoint, B; does not use the assumption that alpha=sin(alpha) when doing its computations
Compute_Turning_and_ Nonturning_Distances	generic procedure	Computes missile turning distance projected onto current course segment, AB, and the missile non-turning distance measured along the next course segment, BC
Turn_Test_Operations	generic package	Contains operations to determine if the missile should be in turning or nonturning flight

3.6.6.1.9 PART DESIGN

3.6.6.1.9.1 STEERING_VECTOR_OPERATIONS (CATALOG #P100-0)

This package contains operations to do the following:

- o Initialize the waypoint steering vectors when supplied with the latitude and longitude of the past, current, and next waypoints
- o Update the waypoint steering vectors when supplied with the latitude and longitude of the "new" waypoint, C.

The waypoint steering vectors for a course segment, extending from waypoint A to waypoint B, are the segment unit normal vector (UN_B) and the segment unit tangent vector (UT_B).

This part makes the assumption that alphasin(alpha) when calculating Segment_-BC_Distance.

3.6.6.1.9.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Initialize	R170
Update	R171

3.6.6.1.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors
Earth_ Distances	floating point type	Data type used to define the radius of the Earth
Earth_Positions	floating point type	Data type used to define latitude and longitude measurements
Segment_ Distances	floating point type	Data type used to defined waypoint segment distances
Sin_Cos_Ratio	floating point type	Data type used to define results of a sine or cosine function
Unit_Vectors	array	Array of "Sin_Cos_Ratio" dimensioned by Indices

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_Distances	N/A	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the operation: Earth_Distances * Sin_Cos_Ratio => Segment_Distances
"/"	function	Operator defining the operation: Unit_Vectors / Sin_Cos_Ratio => Unit_Vectors
Cross_Product	procedure	Cross product function
Vector_Length	function	Calculates the length of a vector
Sin_Cos	procedure	Calculates the sine and cosine of an input value

3.6.6.1.9.1.3 LOCAL ENTITIES

Data structures:

This part maintains the following data:
1) Unit radial vector to waypoint B
2) Unit radial vector to waypoint C

Packages:

The following parts will be with'd by this part's body:
1. Geometric_Parts (P684)

Subprograms:

The following parts, contained in the Geometric_Parts TLCSC, will be instantiated in this part's body and used by the units in this LLCSC:
1. Compute_Unit_Radial_Vector
2. Compute_Segment_and_Unit_Normal_Vector

3.6.6.1.9.1.4 INTERRUPTS

None.

3.6.6.1.9.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Waypoint_Steering;
with Basic_Data_Types; use Basic_Data_Types;
with Coordinate_Vector_Matrix_Algebra;
with WGS72_Ellipsoid_Engineering_Data;
...
   package WPS   renames Waypoint_Steering;
   package BDT   renames Basic_Data_Types;
   package CVMA  renames Coordinate_Vector_Matrix_Algebra;
   package WGS72 renames WGS72_Ellipsoid_Engineering_Data;
   ...
   type Indices is (X, Y, Z);
```

```
...
package Unit_Vector_Opns is new CVMA.Vector_Operations ...
use Unit_Vector_Opns;
subtype Unit_Vectors is Unit_Vector_Opns.Vectors;
...
function Cross_Prod is new CVMA.Cross_Product ...
...
package Unit_Vector_Scalar_Opns is new
            CVMA.Vector_Scalar_Operations ...
...
package Steering_Vector_Opns is new
            WPS.Steering_Vector_Operations
              (Indices           => Indices,
               Earth_Distances   => BDT.Meters,
               Earth_Positions   => Earth_Position_Radians,
               Segment_Distances => BDT.Meters,
               Sin_Cos_Ratio     => BDT.Trig.Sin_Cos_Ratio,
               Unit_Vectors      => Unit_Vectors,
               Earth_Radius      => WGS72.Semimajor_Axis,
               "/"               => Unit_Vector_Scalar_Opns."/",
               Cross_Product     => Cross_Prod,
               Sin_Cos           => BDT.Trig.Sin_Cos);
...
Lat_A       : BDT.Earth_Position_Radians;
Lat_B       : BDT.Earth_Position_Radians;
Lat_C       : BDT.Earth_Position_Radians;
Long_A      : BDT.Earth_Position_Radians;
Long_B      : BDT.Earth_Position_Radians;
Long_C      : BDT.Earth_Position_Radians;
UN_B        : Unit_Vectors;
UN_C        : Unit_Vectors;
UT_B        : Unit_Vectors;
UT_C        : Unit_Vectors;
BC_Distance : BDT.Meters;
...
   begin
      ...
         Steering_Vector_Opns.Initialize
           (Waypoint_A_Lat      => Lat_A,
            Waypoint_A_Long     => Long_A,
            Waypoint_B_Lat      => Lat_B,
            Waypoint_B_Long     => Long_B,
            Waypoint_C_Lat      => Lat_C,
            Waypoint_C_Long     => Long_C,
            Unit_Normal_B       => UN_B,
            Unit_Normal_C       => UN_C,
            Unit_Tangent_B      => UT_B,
            Unit_Tangent_C      => UT_C,
            Segment_BC_Distance => BC_Distance);
```

3.6.6.1.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.6.1.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Initialize	procedure	Initializes the waypoint steering vectors when supplied with the latitude and longitude of the last, current, and next waypoints
Update	procedure	Updates the waypoint steering vectors when supplied with the latitude and longitude of a "new" next waypoint, C

3.6.6.1.9.1.8 PART DESIGN

None.

3.6.6.1.9.2 STEERING_VECTOR_OPERATIONS_WITH_ARCSIN (CATALOG #P1047-0)

This package contains operations to do the following:
- o Initialize the waypoint steering vectors when supplied with the latitude and longitude of the past, current, and next waypoints
- o Update the waypoint steering vectors when supplied with the latitude and longitude of the "new" waypoint, C.

The waypoint steering vectors for a course segment, extending from waypoint A to waypoint B, are the segment unit normal vector (UN_B) and the segment unit tangent vector (UT_B).

This part does not make the assumption that alphasin(alpha) when computing Segment_BC_Distance.

3.6.6.1.9.2.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Initialize	
Update	

3.6.6.1.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors
Earth_Distances	floating point type	Data type used to define the radius of the Earth
Earth_Positions	floating point type	Data type used to define latitude and longitude measurements
Segment_Distances	floating point type	Data type used to defined waypoint segment distances
Sin_Cos_Ratio	floating point type	Data type used to define results of a sine or cosine function
Unit_Vectors	array	Array of "Sin_Cos_Ratio" dimensioned by Indices

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_Distances	N/A	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the operation: Earth_Distances * Sin_Cos_Ratio => Segment_Distances
"/"	function	Operator defining the operation: Unit_Vectors / Sin_Cos_Ratio => Unit_Vectors
Arcsin	function	Arcsine function
Cross_Product	procedure	Cross product function
Vector_Length	function	Calculates the length of a vector
Sin_Cos	procedure	Calculates the sine and cosine of an input value

3.6.6.1.9.2.3 LOCAL ENTITIES

Data structures:

This part maintains the following data:

1) Unit radial vector to waypoint B
2) Unit radial vector to waypoint C

Packages:

The following parts will be with'd by this part's body:
1. Geometric_Parts (P684)

Subprograms:

The following parts, contained in the Geometric_Parts TLCSC, will be instantiated in this part's body and used by the units in this LLCSC:
1. Compute_Unit_Radial_Vector
2. Compute_Segment_and_Unit_Normal_Vector_with_Arcsin

3.6.6.1.9.2.4 INTERRUPTS

None.

3.6.6.1.9.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Waypoint_Steering;
with Basic_Data_Types; use Basic_Data_Types;
with Coordinate_Vector_Matrix_Algebra;
with WGS72_Ellipsoid_Engineering_Data;
...
   package WPS   renames Waypoint_Steering;
   package BDT   renames Basic_Data_Types;
   package CVMA  renames Coordinate_Vector_Matrix_Algebra;
   package WGS72 renames WGS72_Ellipsoid_Engineering_Data;
   ...
   type Indices is (X, Y, Z);
   ...
   package Unit_Vector_Opns is new CVMA.Vector_Operations ...
   use Unit_Vector_Opns;
   subtype Unit_Vectors is Unit_Vector_Opns.Vectors;
   ...
   function Cross_Prod is new CVMA.Cross_Product ...
   ...
   package Unit_Vector_Scalar_Opns is new
              CVMA.Vector_Scalar_Operations ...
   ...
   package Steering_Vector_Opns is new
              WPS.Steering_Vector_Operatio  _with_Arcsin
                 (Indices           => Indices,
                  Earth_Distances   => BDT.Meters,
                  Earth_Positions   => Earth_Position_Radians,
                  Radians           => BDT.Trig.Radians,
                  Segment_Distances => BDT.Meters,
                  Sin_Cos_Ratio     => BDT.Trig.Sin_Cos_Ratio,
                  Unit_Vectors      => Unit_Vectors,
                  Earth_Radius      => WGS72.Semimajor_Axis,
```

```
                 "/"                => Unit_Vector_Scalar_Opns."/",
                 Cross_Product      => Cross_Prod,
                 Sin_Cos            => BDT.Trig.Sin_Cos);
   ...
   Lat_A       : BDT.Earth_Position_Radians;
   Lat_B       : BDT.Earth_Position_Radians;
   Lat_C       : BDT.Earth_Position_Radians;
   Long_A      : BDT.Earth_Position_Radians;
   Long_B      : BDT.Earth_Position_Radians;
   Long_C      : BDT.Earth_Position_Radians;
   UN_B        : Unit_Vectors;
   UN_C        : Unit_Vectors;
   UT_B        : Unit_Vectors;
   UT_C        : Unit_Vectors;
   BC_Distance : BDT.Meters;
   ...
     begin
       ...
         Steering_Vector_Opns.Initialize
            (Waypoint_A_Lat      => Lat_A,
             Waypoint_A_Long     => Long_A,
             Waypoint_B_Lat      => Lat_B,
             Waypoint_B_Long     => Long_B,
             Waypoint_C_Lat      => Lat_C,
             Waypoint_C_Long     => Long_C,
             Unit_Normal_B       => UN_B,
             Unit_Normal_C       => UN_C,
             Unit_Tangent_B      => UT_B,
             Unit_Tangent_C      => UT_C,
             Segment_BC_Distance => BC_Distance);
```

3.6.6.1.9.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.6.1.9.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Initialize	procedure	Initializes the waypoint steering vectors when supplied with the latitude and longitude of the last, current, and next waypoints
Update	procedure	Updates the waypoint steering vectors when supplied with the latitude and longitude of a "new" next waypoint, C

3.6.6.1.9.2.8 PART DESIGN

None.

3.6.6.1.9.3 COMPUTE_TURN_ANGLE_AND_DIRECTION (CATALOG #P101-0)

Using the waypoint steering vectors, this part computes the tangent of one-half the turn angle along with the turn direction.

3.6.6.1.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R172.

3.6.6.1.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Unit_Vectors	private	One-dimensional, three-element arrays defining the waypoint steering vectors
Sin_Cos_Ratio	floating point type	Data type of results of sine/cosine operations
Tan_Ratio	floating point type	Data type of results of tangent operations

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"+"	function	Addition operator defining the operation: Tan_Ratio + Sin_Cos_Ratio => Tan_Ratio
"/"	function	Division operator defining the operation: Sin_Cos_Ratio / Tan_Ratio => Tan_Ratio
Dot_Product	function	Calculates the dot product of two Unit_Vectors

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_Normal_C	Unit_Vectors	in	Segment BC unit normal vector with x, y, and z components
Unit_Tangent_B	Unit_Vectors	in	Segment AB unit tangent vector with x, y, and z components
Unit_Tangent_C	Unit_Vectors	in	Segment BC unit tangent vector with x, y, and z components
Tan_of_One_Half_Turn_Angle	Tan_Ratio	out	Tangent of one-half the angle between the current course segment and the next course segment
Turn_Direction	Turning_Directions	out	Indicates if missile is to make a right- or left-hand turn

The unit vectors required by this part may be calculated using the routines contained in the Waypoint_Steering.Steering_Vector_Operations package.

3.6.6.1.9.3.3 INTERRUPTS

None.

3.6.6.1.9.3.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Waypoint_Steering;
with Basic_Data_Types; use Basic_Data_Types;
with Coordinate_Vector_Matrix_Algebra;
with General_Purpose_Math;
...
   package WPS    renames Waypoint_Steering;
   package BDT    renames Basic_Data_Types;
   package CVMA   renames Coordinate_Vector_Matrix_Algebra;
   package GPMath renames General_Purpose_Math;
   ...
   type Indices is (X, Y, Z);
   ...
   package Unit_Vector_Opns is new CVMA.Vector_Operations ...
   use Unit_Vector_Opns;
   subtype Unit_Vectors is Unit_Vector_Opns.Vectors;
   ...
   package Unit_Vector_Scalar_Opns is new
             CVMA.Vector_Scalar_Operations ...
   ...
   procedure Comp_Turn_Angle_and_Direction is new
               WPS.Compute_Turn_Angle_and_Direction
                  (Unit_Vectors  => Unit_Vectors,
                   Sin_Cos_Ratio => BDT.Trig.Sin_Cos_Ratio,
                   Tan_Ratio     => BDT.Trig.Tan_Ratio,
                   Dot_Product   => Unit_Vector_Opns.Dot_Product);
   ...
```

```
   UN_C            : Unit_Vectors;
   UT_B            : Unit_Vectors;
   UT_C            : Unit_Vectors;
   Tan_Value       : Tan_Ratio;
   Turn_Direction  : WPS.Turning_Directions;
   ...
      begin
         ...
         Comp_Turn_Angle_and_Direction
            (Unit_Normal_C                 => UN_C,
             Unit_Tangent_B                => UT_B,
             Unit_Tangent_C                => UT_C,
             Tan_of_One_Half_Turn_Angle => Tan_Value,
             Turn_Direction                => Turn_Direction);
         ...
```

3.6.6.1.9.3.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.6.1.9.3.6 DECOMPOSITION

None.

3.6.6.1.9.4 CROSSTRACK_AND_HEADING_ERROR_OPERATIONS (CATALOG #P102-0)

This part contains the routines required to compute the crosstrack and heading errors for a missile in turning or nonturning flight.

3.6.6.1.9.4.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Compute_When_Turning	R173
Compute_When_Not_Turning	R175
Compute	R174

3.6.6.1.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Navigation_Indices	discrete type	Data type used to dimension Velocity_ Vectors
Unit_Indices	discrete type	Data type used to dimension Unit_Vectors
Angles	floating point type	Data type of angular measurements
Earth_Distances	floating point type	Data type of distance measurements used to defined Earth radius
Segment_Distances	floating point type	Data type used to measure segments of the flight
Sin_Cos_Ratio	floating point type	Data type of results of sine/cosine operations
Tan_Ratio	floating point type	Data type of tangent operations
Velocities	floating point type	Data type of velocity measurements
Unit_Vectors	array	Array, dimensioned by Unit_Indices, of Sin_Cos_Ratio
Velocity_Vectors	array	Array, dimensioned by Navigation_ Indices, of Velocities

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
E(ast)	Navigation_ Indices	'FIRST	Used to access first element of arrays dimensioned by Navigation_ Indices
N(orth)	Navigation_ Indices	'SUCC(E)	Used to access second element of arrays dimensioned by Navigation_ Indices
U(p)	Navigation_ Indices	'LAST	Used to access last element of arrays dimensioned by Navigation_ Indices
X	Unit_Indices Indices	'FIRST	Used to access first element of arrays dimensioned by Unit_ Indices
Y	Unit_Indices Indices	'SUCC(X)	Used to access second element of arrays dimensioned by Unit_ Indices
Z	Unit_Indices Indices	'LAST	Used to access last element of arrays dimensioned by Unit_ Indices
Earth_Radius	Earth_ Distances	n/a	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Earth_Distances => Segment_Distances
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Segment_Distances => Segment_Distances
"*"	function	Multiplication operator defining the operation: Segment_Distances * INTEGER => Segment_Distances
"*"	function	Multiplication operator defining the operation: INTEGER * Sin_Cos_Ratio => Sin_Cos_Ratio
"*"	function	Multiplication operator defining the operation: Distances * Velocities => Tan_Ratio
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Velocities => Velocities
"/"	function	Division operator defining the operation: Velocities / Velocities => Tan_Ratio
Dot_Product	function	Dot product function
Sqrt	function	Square root function
Arctan	function	Arctangent function

3.6.6.1.9.4.3 LOCAL ENTITIES

None.

3.6.6.1.9.4.4 INTERRUPTS

None.

3.6.6.1.9.4.5 TIMING AND SEQUENCING

The following units may be used to calculate the input values required by units in this package:

- o Waypoint_Steering.Distance_to_Current_Waypoint
- o Waypoint_Steering.Steering_Vector_Operations.Initialize and Update
- o Waypoint_Steering.Compute_Turn_Angle_And_Direction

The following shows a sample usage of this part:

```
with Waypoint_Steering;
with Basic_Data_Types; use Basic_Data_Types;
with General_Purpose_Math;
with Coordinate_Vector_Algebra;
with WGS72_Ellipsoid_Engineering_Data;
...
   package WPS    renames Waypoint_Steering;
```

```
package BDT    renames Basic_Data_Types;
package GPMAth renames General_Purpose_Math;
package CVMA   renames Coordinate_Vector_Algebra;
package WGS72  renames WGS72_Ellipsoid_Engineering_Data;
...
type Navigation_Indices is (E, N, U);
type Unit_Indices       is (X, Y, Z);
...
package Unit_VOpns     is new CVMA.Vector_Operations ...
package Velocity_VOpns is new CVMA.Vector_Operations ...
package Sqrt_Pkg       is new GPMath.Square_Root ...
...
subtype Unit_Vectors     is Unit_VOpns.Vectors;
subtype Velocity_Vectors is Velocity_VOpns.Vectors;
...
package Crosstrack_and_Hdg_Error_Opns is new
          WPS.Crosstrack_and_Heading_Error_Operations
             (Navigation_Indices => Navigation_Indices,
              Unit_Indices       => Unit_Indices,
              Angles             => BDT.Trig.Radians,
              Earth_Distances    => BDT.Meters,
              Segment_Distances  => BDT.Meters,
              Sin_Cos_Ratio      => BDT.Trig.Sin_Cos_Ratio,
              Tan_Ratio          => BDT.Trig.Tan_Ratio,
              Velocities         => BDT.Meters_per_Second,
              Unit_Vectors       => Unit_Vectors      ,
              Velocity_Vectors   => Velocity_Vectors  ,
              Earth_Radius       => WGS72.Semimajor_Axis,
              ...
              Dot_Product        => Unit_VOpns.Dot_Product,
              Sqrt               => Sqrt_Pkg.Sqrt,
              Arctan             => BDT.Trig.Arctan);
...
UR_M              : Unit_Vectors;
UN_B              : Unit_Vectors;
Grnd_Vel          : Velocity_Vectors;
Crosstrack_Error  : BDT.Meters;
Heading_Error     : BDT.Radians;
...
   begin
     ...
     Crosstrack_and_Hdg_Error_Opns.Compute_When_Not_Turning
        (Unit_Radial_M    => UR_M,
         Unit_Normal_B    => UN_B,
         Ground_Velocity  => Grnd_Vel,
         Crosstrack_Error => Crosstrack_Error,
         Heading_Error    => Heading_Error);
     ...
```

3.6.6.1.9.4.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.6.1.9.4.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Compute_When_Turning	procedure	Computes crosstrack and heading error for a missile in turning flight
Compute_When_Not_Turning	procedure	Computes crosstrack and heading error for a missile in nonturning flight
Compute	procedure	Computes crosstrack and heading error for a missile

3.6.6.1.9.4.8 PART DESIGN

None.

3.6.6.1.9.5 DISTANCE_TO_CURRENT_WAYPOINT (CATALOG #P103-0)

This part computes the distance from the missile's position to the current waypoint, B.

This part uses the assumption that alpha=sin(alpha) for its calculations.

3.6.6.1.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R176.

3.6.6.1.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Unit_Vectors	private	One dimensional, three-element array of Sin_Cos_Ratio
Sin_Cos_Ratio	floating point type	Results of sine/cosine operations
Tan_Ratio	floating point type	Results of tangent operation
Earth_Distances	floating point type	Data type of distance measurements involving the radius of the Earth
Segment_Distances	floating point type	Data type of distance measurements involving navigation segments

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_Distances	N/A	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Dot_Product	function	Computes the dot product of two unit vectors
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Earth_Distances => Segment_Distances

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_Radial_M	Unit_Vectors	in	Unit radial vector to the missile extending outward from the origin of the Earth-centered reference frame
Unit_Tangent_B	Unit_Vectors	in	Segment AB unit tangent vector

The unit vectors may be calculated using the Waypoint_Steering. Steering_-Vector_Operations routines.

3.6.6.1.9.5.3 INTERRUPTS

None.

3.6.6.1.9.5.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Waypoint_Steering;
with Basic_Data_Types; use Basic_Data_Types;
with Coordinate_Vector_Algebra;
with WGS72_Ellipsoid_Engineering_Data;
...
   package WPS    renames Waypoint_Steering;
   package BDT    renames Basic_Data_Types;
   package CVMA   renames Coordinate_Vector_Algebra;
   package WGS72  renames WGS72_Ellipsoid_Engineering_Data;
   ...
   type Unit_Indices    is (X, Y, Z);
   ...
   package  Unit_VOpns  is new CVMA.Vector_Operations ...
   ...
   subtype Unit_Vectors is Unit_VOpns.Vectors;
   ...
   function Dist_to_Current_Waypoint is new
               WPS.Distance_to_Current_Waypoint
                  (Unit_Vectors       => Unit_Vectors,
                   Sin_Cos_Ratio      => BDT.Trig.Sin_Cos_Ratio,
                   Tan_Ratio          => BDT.Trig.Tan_Ratio,
                   Earth_Distances    => BDT.Meters,
                   Segment_Distances  => BDT.Meters,
                   Earth_Radius       => WGS72.Semimajor_Axis,
                   Dot_Product        => Unit_VOpns.Dot_Product);
   ...
   UR_M        : Unit_Vectors;
   UT_B        : Unit_Vectors;
   Dist_to_B   : BDT.Meters;
   ...
      begin
         ...
         Dist_to_B := Dist_to_Current_Waypoint
                         (Unit_Radial_M  => UR_M,
                          Unit_Tangent_B => UT_B);
         ...
```

3.6.6.1.9.5.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.6.1.9.5.6 DECOMPOSITION

None.

3.6.6.1.9.6 DISTANCE_TO_CURRENT_WAYPOINT_WITH_ARCSIN

This part computes the distance from the missile's position to the current waypoint, B.

This part does not use the assumpting that alpha=sin(alpha) when doing its computations.

3.6.6.1.9.6.1 REQUIREMENTS ALLOCATION

None.

3.6.6.1.9.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Unit_Vectors	private	One dimensional, three-element array of Sin_Cos_Ratio
Sin_Cos_Ratio	floating point type	Results of sine/cosine operations
Tan_Ratio	floating point type	Results of tangent operation
Earth_ Distances	floating point type	Data type of distance measurements involving the radius of the Earth
Segment_ Distances	floating point type	Data type of distance measurements involving navigation segments

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_Distances	N/A	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Arcsin	function	Arcsine function (must return radians, NOT degrees)
Dot_Product	function	Computes the dot product of two unit vectors
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Earth_Distances => Segment_Distances

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_ Radial_M	Unit_Vectors	in	Unit radial vector to the missile extending outward from the origin of the Earth-centered reference frame
Unit_ Tangent_B	Unit_Vectors	in	Segment AB unit tangent vector

The unit vectors may be calculated using the Waypoint_Steering. Steering_-Vector_Operations routines.

3.6.6.1.9.6.3 INTERRUPTS

None.

3.6.6.1.9.6.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Waypoint_Steering;
with Basic_Data_Types; use Basic_Data_Types;
with Coordinate_Vector_Algebra;
with WGS72_Ellipsoid_Engineering_Data;
...
   package WPS    renames Waypoint_Steering;
   package BDT    renames Basic_Data_Types;
   package CVMA   renames Coordinate_Vector_Algebra;
   package WGS72  renames WGS72_Ellipsoid_Engineering_Data;
   ...
   type Unit_Indices    is (X, Y, Z);
   ...
   package  Unit_VOpns  is new CVMA.Vector_Operations ...
   ...
   subtype Unit_Vectors is Unit_VOpns.Vectors;
   ...
   function Dist_to_Current_Waypoint is new
               WPS.Distance_to_Current_Waypoint_with_Arcsin
```

```
                    (Unit_Vectors      => Unit_Vectors,
                     Sin_Cos_Ratio     => BDT.Trig.Sin_Cos_Ratio,
                     Tan_Ratio         => BDT.Trig.Tan_Ratio,
                     Earth_Distances   => BDT.Meters,
                     Segment_Distances => BDT.Meters,
                     Earth_Radius      => WGS72.Semimajor_Axis,
                     Dot_Product       => Unit_VOpns.Dot_Product);
   ...
   UR_M        : Unit_Vectors;
   UT_B        : Unit_Vectors;
   Dist_to_B   : BDT.Meters;
   ...
      begin
         ...
         Dist_to_B := Dist_to_Current_Waypoint
                         (Unit_Radial_M  => UR_M,
                          Unit_Tangent_B => UT_B);
         ...
```

3.6.6.1.9.6.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.6.1.9.6.6 DECOMPOSITION

None.

3.6.6.1.9.7 COMPUTE_TURNING_AND_NONTURNING_DISTANCES (CATALOG #P104-0)

This part computes the missile turning distance projected onto the current course segment, AB, and the missile nonturning distance measured along the next course segment, BC.

3.6.6.1.9.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R177.

3.6.6.1.9.7.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Distances	floating point type	Data type of distance measurements
Tan_Ratio	floating point type	Data type of results of tangent operation

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Distances * Tan_Ratio => Distances

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Tan_of_One_Half_Turn_Angle	Tan_Ratio	in	Tangent of 1/2 the angle between current course segment and next course segment
Segment_BC_Distance	Distances	in	Great circle arc length between way-points B and C
Turn_Radius	Distances	in	Desired missile turn radius
Turning_Distance	Distances	out	Distance from the point of tangency of the turn circle and the current course segment AB to the current waypoint, B
Nonturning_Distance	Distances	[illegible]	Distance from the point of tangency of the turn circle and the next course segment BC to the next waypoint, C

These input values may be calculated using the following parts:
- o Waypoint_Steering.Compute_Turn_Angle_And_Direction
- o Waypoint_Steering.Steering_Vector_Operations.Initialize and Update

3.6.6.1.9.7.3 INTERRUPTS

None.

3.6.6.1.9.7.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Waypoint_Steering;
```

```
with Basic_Data_Types; use Basic_Data_Types;
...
   package WPS renames Waypoint_Steering;
   package BDT renames Basic_Data_Types;
   ...
   procedure Comp_Turn_And_Nonturn_Dist
                 WPS.Compute_Turning_and_Nonturning_Distances
                    (Distances => BDT.Meters,
                     Tan_Ratio => BDT.Trig.Tan_Ratio);
   ...
   Tan_Value     : BDT.Trig.Tan_Ratio;
   BC_Dist       : BDT.Meters;
   Turn_Rad      : BDT.Meters;
   Turn_Dist     : BDT.Meters;
   Nonturn_Dist  : BDT.Meters;
   ...
      begin
        ...
          Comp_Turn_And_Nonturn_Dist is new
             (Tan_of_One_Half_Turn_Angle => Tan_Value,
              Segment_BC_Distance        => BC_Dist,
              Turn_Radius                => Turn_Rad,
              Turning_Distance           => Turn_Dist,
              Nonturning_Distance        => Nonturn_Dist);
        ...
```

3.6.6.1.9.7.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.6.1.9.7.6 DECOMPOSITION

None.

3.6.6.1.9.8 TURN_TEST_OPERATIONS (CATALOG #P105-0)

This part contains the operations required to determine if a missile should be in turning or nonturning flight.

3.6.6.1.9.8.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Stop_Test	R178
Start_Test	R179

3.6.6.1.9.8.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Distances	floating	Data type of distance measurements

3.6.6.1.9.8.3 LOCAL ENTITIES

None.

3.6.6.1.9.8.4 INTERRUPTS

None.

3.6.6.1.9.8.5 TIMING AND SEQUENCING

The following parts may be used to compute the input values required by the units in this package:

- o Waypoint_Steering.Distance_to_Current_Waypoint
- o Waypoint_Steering.Compute_Turning_and_Nonturning_Distances

The following shows a sample usage of this part:

```
with Basic_Data_Types; use Basic_Data_Types;
with Waypoint_Steering;
...
   package BDT renames Basic_Data_Types;
   package WPS renames Waypoint_Steering;
   ...
   package Turn_Test_Opns is new
              WPS.Turn_Test_Operations
                 (Distances => BDT.Meters);
   ...
   Dist_to_B    : BDT.Meters;
   Nonturn_Dist : BDT.Meters;
   Lead_Dist    : BDT.Meters;
   Turn_Stat    : WPS.Turning_Statuses;
   ...
     begin
       ...
       Turn_Stat := Turn_Test_Opns.Stop_Test
                        (Distance_to_B       => Dist_to_B,
                         Nonturning_Distance => Nonturn_Dist,
                         Lead_Distance       => Lead_Dist);
       ...
```

3.6.6.1.9.8.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.6.1.9.8.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Stop_Test	function	Indicates if the missile should be in nonturning flight
Start_Test	function	Indicates if the missile should be in turning flight

3.6.6.1.9.8.8 PART DESIGN

None.

```
package Waypoint_Steering is

-- -----------------------
--   --set up required types-
--   ------------------------

   type Turning_Directions is (Left_Turn, Right_Turn);
   type Turning_Statuses   is (Turning,   Not_Turning);

pragma PAGE;
   generic
      type Indices            is (<>);
      type Earth_Distances    is digits <>;
      type Earth_Positions    is digits <>;
      type Segment_Distances is digits <>;
      type Sin_Cos_Ratio      is digits <>;
      type Unit_Vectors       is array (Indices) of Sin_Cos_Ratio;
      Earth_Radius            : in Earth_Distances;
      with function "*" (Left  : Earth_Distances;
                         Right : Sin_Cos_Ratio) return Segment_Distances is <>;
      with function "/" (Left  : Unit_Vectors;
                         Right : Sin_Cos_Ratio) return Unit_Vectors is <>;
      with function Cross_Product (Left  : Unit_Vectors;
                                   Right : Unit_Vectors)
                                  return Unit_Vectors is <>;
      with function Vector_Length (Input : Unit_Vectors)
                                  return Sin_Cos_Ratio is <>;
      with procedure Sin_Cos (Input  : in  Earth_Positions;
                              Sine   : out Sin_Cos_Ratio;
                              Cosine : out Sin_Cos_Ratio) is <>;
   package Steering_Vector_Operations is

      procedure Initialize
                   (Waypoint_A_Lat      : in     Earth_Positions;
                    Waypoint_A_Long     : in     Earth_Positions;
                    Waypoint_B_Lat      : in     Earth_Positions;
                    Waypoint_B_Long     : in     Earth_Positions;
                    Waypoint_C_Lat      : in     Earth_Positions;
                    Waypoint_C_Long     : in     Earth_Positions;
                    Unit_Normal_B       :    out Unit_Vectors;
                    Unit_Normal_C       :    out Unit_Vectors;
                    Unit_Tangent_B      :    out Unit_Vectors;
                    Unit_Tangent_C      :    out Unit_Vectors;
                    Segment_Bc_Distance :    out Segment_Distances);

      procedure Update
                   (Waypoint_C_Lat      : in     Earth_Positions;
                    Waypoint_C_Long     : in     Earth_Positions;
                    Unit_Normal_B       :    out Unit_Vectors;
                    Unit_Normal_C       : in out Unit_Vectors;
                    Unit_Tangent_B      :    out Unit_Vectors;
                    Unit_Tangent_C      : in out Unit_Vectors;
                    Segment_Bc_Distance :    out Segment_Distances);

   end Steering_Vector_Operations;

pragma PAGE;
```

```
   generic
      type Indices            is (<>);
      type Earth_Distances    is digits <>;
      type Earth_Positions    is digits <>;
      type Radians            is digits <>;
      type Segment_Distances is digits <>;
      type Sin_Cos_Ratio      is digits <>;
      type Unit_Vectors       is array (Indices) of Sin_Cos_Ratio;
      Earth_Radius            : in Earth_Distances;
      with function "*" (Left  : Earth_Distances;
                         Right : Radians) return Segment_Distances is <>;
      with function "/" (Left  : Unit_Vectors;
                         Right : Sin_Cos_Ratio) return Unit_Vectors is <>;
      with function Arcsin (Input : Sin_Cos_Ratio) return Radians is <>;
      with function Cross_Product (Left  : Unit_Vectors;
                                   Right : Unit_Vectors)
                                  return Unit_Vectors is <>;
      with function Vector_Length (Input : Unit_Vectors)
                                  return Sin_Cos_Ratio is <>;
      with procedure Sin_Cos (Input  : in  Earth_Positions;
                              Sine   : out Sin_Cos_Ratio;
                              Cosine : out Sin_Cos_Ratio) is <>;
   package Steering_Vector_Operations_With_Arcsin is

      procedure Initialize
                  (Waypoint_A_Lat      : in     Earth_Positions;
                   Waypoint_A_Long     : in     Earth_Positions;
                   Waypoint_B_Lat      : in     Earth_Positions;
                   Waypoint_B_Long     : in     Earth_Positions;
                   Waypoint_C_Lat      : in     Earth_Positions;
                   Waypoint_C_Long     : in     Earth_Positions;
                   Unit_Normal_B       :    out Unit_Vectors;
                   Unit_Normal_C       :    out Unit_Vectors;
                   Unit_Tangent_B      :    out Unit_Vectors;
                   Unit_Tangent_C      :    out Unit_Vectors;
                   Segment_Bc_Distance :    out Segment_Distances);

      procedure Update
                  (Waypoint_C_Lat      : in     Earth_Positions;
                   Waypoint_C_Long     : in     Earth_Positions;
                   Unit_Normal_B       :    out Unit_Vectors;
                   Unit_Normal_C       : in out Unit_Vectors;
                   Unit_Tangent_B      :    out Unit_Vectors;
                   Unit_Tangent_C      : in out Unit_Vectors;
                   Segment_Bc_Distance :    out Segment_Distances);

   end Steering_Vector_Operations_With_Arcsin;

pragma PAGE;
   generic
      type Unit_Vectors  is private;
      type Sin_Cos_Ratio is digits <>;
      type Tan_Ratio     is digits <>;
      with function "+" (Left  : Tan_Ratio;
                         Right : Sin_Cos_Ratio) return Tan_Ratio is <>;
      with function "/" (Left  : Sin_Cos_Ratio;
                         Right : Tan_Ratio) return Tan_Ratio is <>;
```

```
      with function Dot_Product (Left  : Unit_Vectors;
                                 Right : Unit_Vectors)
                                return Sin_Cos_Ratio is <>;
   procedure Compute_Turn_Angle_And_Direction
                (Unit_Normal_C               : in     Unit_Vectors;
                 Unit_Tangent_B              : in     Unit_Vectors;
                 Unit_Tangent_C              : in     Unit_Vectors;
                 Tan_Of_One_Half_Turn_Angle  :    out Tan_Ratio;
                 Turn_Direction              :    out Turning_Directions);

pragma PAGE;
   generic
      type Navigation_Indices is (<>);
      type Unit_Indices       is (<>);
      type Angles             is digits <>;
      type Earth_Distances    is digits <>;
      type Segment_Distances  is digits <>;
      type Sin_Cos_Ratio      is digits <>;
      type Tan_Ratio          is digits <>;
      type Velocities         is digits <>;
      type Unit_Vectors       is array (Unit_Indices) of Sin_Cos_Ratio;
      type Velocity_Vectors   is array (Navigation_Indices) of Velocities;
      E            : in Navigation_Indices := Navigation_Indices'FIRST;
      N            : in Navigation_Indices := Navigation_Indices'SUCC(E);
      U            : in Navigation_Indices := Navigation_Indices'LAST;
      X            : in Unit_Indices       := Unit_Indices'FIRST;
      Y            : in Unit_Indices       := Unit_Indices'SUCC(X);
      Z            : in Unit_Indices       := Unit_Indices'LAST;
      Earth_Radius : in Earth_Distances;
      with function "*" (Left  : Sin_Cos_Ratio;
                         Right : Earth_Distances)
                                          return Segment_Distances is <>;
      with function "*" (Left  : Sin_Cos_Ratio;
                         Right : Segment_Distances)
                                          return Segment_Distances is <>;
      with function "*" (Left  : Segment_Distances;
                         Right : INTEGER)         return Segment_Distances is <>;
      with function "*" (Left  : INTEGER;
                         Right : Sin_Cos_Ratio)   return Sin_Cos_Ratio is <>;
      with function "*" (Left  : Segment_Distances;
                         Right : Velocities)      return Tan_Ratio is <>;
      with function "*" (Left  : Sin_Cos_Ratio;
                         Right : Velocities)      return Velocities is <>;
      with function "/" (Left  : Velocities;
                         Right : Velocities)      return Tan_Ratio is <>;
      with function Dot_Product (Left  : Unit_Vectors;
                                 Right : Unit_Vectors)
                                return Sin_Cos_Ratio is <>;
      with function Sqrt (Input : Segment_Distances)
                         return Segment_Distances is <>;
      with function Arctan (Input : Tan_Ratio) return Angles is <>;
   package Crosstrack_And_Heading_Error_Operations is

      procedure Compute_When_Turning
                   (Distance_To_B        : in  Segment_Distances;
                    Nonturning_Distance  : in  Segment_Distances;
                    Unit_Radial_M        : in  Unit_Vectors;
```

```
                  Unit_Normal_B       : in  Unit_Vectors;
                  Unit_Tangent_B      : in  Unit_Vectors;
                  Turn_Direction      : in  Turning_Directions;
                  Ground_Velocity     : in  Velocity_Vectors;
                  Turn_Radius         : in  Segment_Distances;
                  Crosstrack_Error    : out Segment_Distances;
                  Heading_Error       : out Angles);

      procedure Compute_When_Not_Turning
                 (Unit_Radial_M    : in  Unit_Vectors;
                  Unit_Normal_B    : in  Unit_Vectors;
                  Ground_Velocity  : in  Velocity_Vectors;
                  Crosstrack_Error : out Segment_Distances;
                  Heading_Error    : out Angles);

      procedure Compute
                 (Distance_To_B         : in  Segment_Distances;
                  Nonturning_Distance   : in  Segment_Distances;
                  Unit_Radial_M         : in  Unit_Vectors;
                  Unit_Normal_B         : in  Unit_Vectors;
                  Unit_Tangent_B        : in  Unit_Vectors;
                  Turn_Direction        : in  Turning_Directions;
                  Turn_Status           : in  Turning_Statuses;
                  Ground_Velocity       : in  Velocity_Vectors;
                  Turn_Radius           : in  Segment_Distances;
                  Crosstrack_Error      : out Segment_Distances;
                  Heading_Error         : out Angles);

   end Crosstrack_And_Heading_Error_Operations;

pragma PAGE;
   generic
      type Unit_Vectors       is private;
      type Sin_Cos_Ratio      is digits <>;
      type Tan_Ratio          is digits <>;
      type Earth_Distances    is digits <>;
      type Segment_Distances  is digits <>;
      Earth_Radius            : in Earth_Distances;
      with function Dot_Product (Left  : Unit_Vectors;
                                 Right : Unit_Vectors)
                                return Sin_Cos_Ratio is <>;
      with function "*" (Left  : Sin_Cos_Ratio;
                         Right : Earth_Distances)
                        return Segment_Distances is <>;
   function Distance_To_Current_Waypoint
            (Unit_Radial_M  : Unit_Vectors;
             Unit_Tangent_B : Unit_Vectors) return Segment_Distances;

pragma PAGE;
   generic
      type Unit_Vectors       is private;
      type Radians            is digits <>;
      type Sin_Cos_Ratio      is digits <>;
      type Tan_Ratio          is digits <>;
      type Earth_Distances    is digits <>;
      type Segment_Distances  is digits <>;
      Earth_Radius            : in Earth_Distances;
```

```
      with function Arcsin (Input : Sin_Cos_Ratio) return Radians is <>;
      with function Dot_Product (Left  : Unit_Vectors;
                                 Right : Unit_Vectors)
                                return Sin_Cos_Ratio is <>;
      with function "*" (Left  : Radians;
                         Right : Earth_Distances)
                        return Segment_Distances is <>;
   function Distance_To_Current_Waypoint_With_Arcsin
               (Unit_Radial_M  : Unit_Vectors;
                Unit_Tangent_B : Unit_Vectors) return Segment_Distances;

pragma PAGE;
   generic
      type Distances is digits <>;
      type Tan_Ratio is digits <>;
      with function "*" (Left  : Distances;
                         Right : Tan_Ratio) return Distances is <>;
   procedure Compute_Turning_And_Nonturning_Distances
               (Tan_Of_One_Half_Turn_Angle : in     Tan_Ratio;
                Segment_Bc_Distance        : in     Distances;
                Turn_Radius                : in     Distances;
                Turning_Distance           :    out Distances;
                Nonturning_Distance        :    out Distances);

pragma PAGE;
   generic
      type Distances is digits <>;
   package Turn_Test_Cperations is

      function Stop_Test
                 (Distance_To_B       : Distances;
                  Nonturning_Distance : Distances;
                  Lead_Distance       : Distances) return Turning_Statuses;

      function Start_Test
                 (Distance_To_B       : Distances;
                  Turning_Distance    : Distances;
                  Lead_Distance       : Distances) return Turning_Statuses;

   end Turn_Test_Operations;

end Waypoint_Steering;
```

(This page left intentionally blank.)

3.6.6.2 AUTOPILOT TLCSC (CATALOG #P301-0)

This package provides operational parts to perform autopilot functions. Each part is designed as an Ada generic package, where the generic parameters will specify the data types of the input and output signals and the values needed by the packages to perform their actual processing.

The method used in developing this design is to offer a package which requires actual subprograms to be used in performing the low level functions. The part user will have constructed these low level subprograms prior to instantiating the package. If the data types used in instantiating the autopilot parts are the same as those used in instantiating the appropriate low level functions, then the low level functions will default; the user need not include those functions in the instantiation of the autopilot part.

There are no exceptions raised at the TLCSC level.

3.6.6.2.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Type	Req. Allocation
Integral Plus Proportional	generic package	R048
Pitch autopilot	generic package	R059
Lateral/Directional autopilot	generic package	R064

3.6.6.2.2 INPUT/OUTPUT

None.

3.6.6.2.3 UTILIZATION OF OTHER ELEMENTS

This package does not with other library units. However, applications using this part should with the following parts:

1. Signal Processing Parts
2. Autopilot Data Types

3.6.6.2.4 LOCAL ENTITIES

None

3.6.6.2.5 INTERRUPTS

None.

3.6.6.2.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Autopilot, Signal_Processing, Autopilot_Data_Types;
procedure User_Application is

   --Instantiate packages from Autopilot

begin

   . . .

end User_Application;
```

3.6.6.2.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.6.2.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Integral plus proportional gain	generic package	This package exports the subprograms to perform the integral plus proportional gain. It also allows for the initialization of the integrator state.
Pitch Autopilot	generic package	Performs the pitch autopilot operations needed to produce a new value for elevator commands. It also initializes the pitch dynamics.
Lateral/ Directional Autopilot	generic package	Performs the lateral/directional auto-pilot operations to produce new values for the aileron and rudder commands. It also initializes the roll and yaw dynamics.

3.6.6.2.9 PART DESIGN

3.6.6.2.9.1 INTEGRAL PLUS PROPORTIONAL GAIN (CATALOG #P302-0)

This part implements the calculations and logic necessary to implement an integral plus proportional gain.

3.6.6.2.9.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Integral_Plus_Proportional_Gain	R048

3.6.6.2.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Input_Signals	generic float	Type of values input to part
Gains	generic float	Type of gain applied to input
Integrated_Signals	generic float	Input signal put through integrator

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Description
Initial_Proportional_Gain	Gains	Initial value of proportional gain applied to input signal

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloads Input_Signals * Gains return Integrated_Signals for proportional gain
Tustin_ Integrate	function	Performs Tustin integrator with limit

3.6.6.2.9.1.3 LOCAL ENTITIES

None.

3.6.6.2.9.1.4 INTERRUPTS

None.

3.6.6.2.9.1.5 TIMING AND SEQUENCING

Sample usage:

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types, Autopilot;
use Autopilot_Data_Types;
procedure USER is
  type Command_Signals is new Autopilot_Data_Types.Roll_Commands;
  type Command_Gains is new
     Autopilot_Data_Types.Degrees_To_Degrees_Per_Second_Gains;
  type Gained_Command_Signals is new
     Autopilot_Data_Types.Feedback_Rates_Degrees;
  package Tustin_Integrator is new
             Signal_Processing.Tustin_Integrator_With_Limit
                (Signals                  => Command_Signals,
                 States                   => Command_Signals,
                 Gained_Signals           => Gained_Command_Signals,
                 Gains                    => Command_Gains,
                 Initial_Tustin_Gain      => 0.0,
                 Initial_Signal_Level     => 0.0,
                 Initial_Time_Interval    => 1.0/64.0,
                 Initial_Signal_Limit     => 5.0);
   use Tustin_Integrator;
   package Command_Integrator is new
             Autopilot.Integral_Plus_Proportional_Gain
                (Input_Signals                => Command_Signals,
                 Gains                        => Command_Gains,
                 Integrated_Signals           => Command_Signals,
                 Proportional_Gain            => 0.1);
                 "*"                      -- Automatically defaults
                 Tustin_Integrate         -- Automatically defaluts
   Command           : Command_Signals;
   Integrated_Signal : Command_Signals;
begin
   Integrated_Signal := Command_Integrator.Integrate (Command);
```

```
   Command_Integrator.Update_Gains
      (New_Proportional_Gain => 0.20);
end USER;
```

3.6.6.2.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.6.2.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Integrate	Function	Performs integral plus proportional gain operations.
Update_Proportional_Gain	Procedure	Stores new values for proportional gain. Integral gain is updated through the Tustin Integrator package.

3.6.6.2.9.1.8 PART DESIGN

None.

3.6.6.2.9.2 PITCH AUTOPILOT (CATALOG #P303-0)

This package implements the functions and procedures necessary to perform a pitch autopilot control loop using classical control theory.

3.6.6.2.9.2.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Pitch Autopilot	R059

3.6.6.2.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Normal_Acceleration_Commands	Generic Float	Type for input commands
Accelerations	Generic Float	Type for acceleration feedbacks
Acceleration_Command_Gains	Generic Float	Gains applied to accelerations in state loop
Acceleration_Gains	Generic Float	Gains applied to filtered acceleration feedback
Pitch_Rates	Generic Float	Type for pitch rate feedback
Pitch_Rate_Gains	Generic Float	Gains applied to filtered pitch rate
Fin_Deflections	Generic Float	Type for Fin Deflection command

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Description
Initial_Integrator_Gain	Acceleration_Command_Gains	Initial gain to Tustin integrator input
Initial_Integrator_Limit	Fin_Deflections	Initial limit on Tustin integrator output
Initial_Acceleration_Gain	Acceleration_Gains	Initial gain to filtered acceleration feedback
Initial_Pitch_Rate_Gain	Pitch_Rate_Gains	Initial gain to filtered pitch rate feedback
Initial_Proportional_Gain	Acceleration_Command_Gains	Initial gain for proportional loop

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Limit	Function	Perform Limiter function (e.g. R202)
Acceleration_ Filter	Function	Performs filter function on Acceleration feedback
Pitch_Rate_Filter	Function	Performs filter function on pitch rate feedback
"*"	Function	Times for applying gain to Normal Acceleration commands
"*"	Function	Times for applying gain to acceleration feedback
"+"	Function	Summer to add input acceleration command and filtered acceleration feedback in autopilot initialize
"-"	Function	Minus for calculating normal acceleration error in integral loop
"*"	Function	Times for applying gain to Pitch Rate feedback

3.6.6.2.9.2.3 LOCAL ENTITIES

Data structures:

Stores values for gains.

Packages:

Instantiates Integral_Plus_Proportional_Gain package (from this Autopilot package) for normal acceleration error Integrator. Also instantiates Tustin Integrator (from Signal_Processing package) as required by Integral_Plus_-Proportional_Gain.

3.6.6.2.9.2.4 INTERRUPTS

None.

3.6.6.2.9.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types, Autopilot;
use Autopilot_Data_Types;
procedure USER is
   package A_D_T renames Autopilot_Data_Types;

   type Command_Signals          is new A_D_T.Acceleration_Commands_FPS2;
   type Acceleration_Feedbacks   is new A_D_T.Acceleration_Feedbacks_GEES;
   type Pitch_Rate_Feedbacks     is new A_D_T.Feedback_Rates_Degrees;
   type Command_Gains            is new A_D_T.FPS2_To_Degrees_Gain;
   type Pitch_Rate_Gains         is new A_D_T.DPS_To_Degrees_Gain;
   type Fin_Deflections          is new A_D_T.Fin_Deflections_Degrees;
   type Acceleration_Filter_Coefficients is new
                                        A_D_T.Acceleration_Feedbacks_FPS2;
```

```
type Pitch_Rate_Filter_Coefficients is new
                                    A_D_T.Feedback_Rate_Degrees;

Normal_Acceleration_Command    : Command_Signals        := 1.0;
Tustin_Gain                    : Command_Gains          := 0.1;
Integrator_Limit               : Fin_Deflections        := 5.0;
Measured_Acceleration_Feedback : Acceleration_Feedbacks := 1.0;
Pitch_Rate_Feedback            : Pitch_Rate_Feedbacks   := 1.0;
Proportional_Gain              : Command_Gains          := 0.1;
Pitch_Rate_Gain                : Pitch_Rate_Gains       := 0.1;
Acceleration_Gain              : Command_Gains          := 0.1;
Elevator_Command               : Gained_Command_Signals := 2.0;

package Command_Limiter is new
           Signal_Processing.Absolute_Limiter_With_Flag
                    (Signal_Type => Fin_Deflections,
                     Initial_Absolute_Limit => 5.0);
use Command_Limiter;             -- to default

package Accelerations_Filter is new
           Signal_Processing.Tustin_Lag_Filter
              (Signal_Type      => Acceleration_Feedbacks,
               Coefficient_Type => Acceleration_Filter_Coefficients,
               Initial_Previous_Input_Signal
                                => Measured_Acceleration_Feedback,
               Coefficient_1    => 0.9,
               Coefficient_2    => 0.1);
function Acceleration_Filter
   (Normal_Acceleration: Acceleration_Feedbacks)
   return Acceleration_Feedbacks renames Acceleration_Filter.Filter;

package Pitch_Rates_Filter is new
           Signal_Processing.Tustin_Lag_Filter
              (Signal_Type      => Pitch_Rate_Feedbacks,
               Coefficient_Type => Pitch_Rate_Filter_Coefficients,
               Initial_Previous_Input_Signal
                                => Pitch_Rate_Feedback,
               Coefficient_1    => 0.9,
               Coefficient_2    => 0.1);
function Pitch_Rate_Filter (Pitch_Rate : Pitch_Rate_Feedbacks)
   returns Pitch_Rate_Feedbacks renames Pitch_Rates_Filter.Filter;

package Pitch_Command is new Pitch_Autopilot
-- Generic actual types
       (Normal_Acceleration_Commands => Command_Signals,
        Accelerations                => Acceleration_Feedbacks,
        Acceleration_Command_Gains   => Command_Gains,
        Acceleration_Gains           => Command_Gains,
        Pitch_Rates                  => Pitch_Rate_Feedbacks,
        Pitch_Rate_Gains             => Pitch_Rate_Gains,
        Fin_Deflections              => Fin_Deflections,
-- Generic actual objects
        Initial_Integrator_Gain      => Tustin_Gain,
        Initial_Integrator_Limit     => Integrator_Limit,
        Initial_Acceleration_Gain    => Acceleration_Gain,
        Initial_Pitch_Rate_Gain      => Pitch_Rate_Gain,
        Initial_Proportional_Gain    => Proportional_Gain
```

```
          -- Limiter and filters default through renamings
          -- all overloaded operators default from Autopilot_Date_Types
                                                                    );

begin

   -- obtain Normal_Acceleration_Command
   -- obtain Acceleration_ and Pitch_Rate_Feedbacks

   Initialize_Pitch_Autopilot
      (Normal_Acceleration_Command  => Normal_Acceleration_Command,
       Measured_Normal_Acceleration => Measured_Acceleration_Feedback,
        Measured_Pitch_Rate          => Pitch_Rate_Feedback,
         Initial_Elevator_Command     => Elevator_Command);

   Elevator_Command := Pitch_Autopilot.Compute_Elevator_Command
      (Normal_Acceleration_Command  => Normal_Acceleration_Command,
       Measured_Normal_Acceleration => Measured_Acceleration_Feedback,
        Measured_Pitch_Rate          => Pitch_Rate_Feedback);

   Pitch_Rate_Gain := 0.2;
   Update_Pitch_Rate_Gain (New_Gain => Pitch_Rate_Gain);

   Acceleration_Gain := 0.2;
   Update_Acceleration_Gain (New_Gain => Acceleration);

   Proportional_Gain := 0.1;
   Update_Proportional_Gain (New_Proportional_Gain => Proportional_Gain);

end USER;
```

3.6.6.2.9.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.6.2.9.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Initialize_Pitch_Autopilot	Procedure	Sets initial state of pitch autopilot loop by initializing integrator state
Compute_Elevator_Command	Function	Accepts acceleration command, measured normal acceleration and pitch rate and uses integrator state to calculate fin deflection
Update_Pitch_Rate_Gain	Procedure	Changes value stored for gain on pitch rate feedback
Update_Acceleration_Gain	Procedure	Changes value stored for gain on acceleration feedback
Update_Integrator_Gain	Procedure	Changes value stored for gain in integrator loop
Update_Integrator_Limit	Procedure	Changes value stored for limit on integrator output
Update_Proportional_Gain_	Procedure	Changes value stored for gain to normal acceleration error in integrator loop

3.6.6.2.9.2.8 PART DESIGN

None.

3.6.6.2.9.3 LATERAL_DIRECTIONAL_AUTOPILOT (CATALOG #P304-0)

This package implements the functions and procedures necessary to perform a lateral directional autopilot control loop using classical control theory.

3.6.6.2.9.3.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Lateral Directional Autopilot	R064

3.6.6.2.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Roll_Commands	Generic Float	Type for input commands from user program
Roll_Attitudes	Generic Float	Type for measured missile roll attitude
Roll_Command_Gains	Generic Float	Gain to Roll commands in integrator loop
Missile_Accelerations	Generic Float	Type for measured lateral acceleration
Acceleration_Gains	Generic Float	Gains applied to measured acceleration
Rudder_Cmd_Roll_Rate_Gains	Generic Float	Gain applied to roll rate feedback for rudder cmd
Gravitational_Accelerations	Generic Float	Type for measured gravitational accel.
Velocities	Generic Float	Type for measured missile velocity
Trig_Value	Generic Float	Type for result of sin function
Fin_Deflections	Generic Float	Type for aileron & rudder commands & limits
Feedback_Rates	Generic Float	Type for measured roll and yaw rates
Feedback_Rate_Gains	Generic Float	Gain applied to yaw and roll rate feedbacks

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Description
Initial_Aileron_Integrator_Gain	Roll_Command_Gains	Gain in aileron integrator loop
Initial_Aileron_Integrator_Limit	Fin_Deflections	Limit on aileron integrator output
Initial_Roll_Command_Proportional_Gain	Roll_Command_Gains	Proportional gain in integrator plus proportional gain loop
Initial_Roll_Rate_Gain_For_Aileron	Feedback_Rate_Gains	Gain to measured roll rate for aileron cmd
Initial_Yaw_Rate_Gain_For_Aileron	Feedback_Rate_Gains	Gain to measured yaw rate for aileron cmd
Initial_Rudder_Integrator_Gain	Acceleration_Gains	Gain in rudder integrator loop
Initial_Rudder_Integrator_Limit	Fin_Deflections	Limit on rudder integrator output
Initial_Yaw_Rate_Gain_For_Rudder	Feedback_Rate_Gains	Gain to measured yaw for rudder command
Initial_Roll_Rate_Gain_For_Rudder	Rudder_Cmd_Roll_Rate_Gains	Gain to measured roll rate for rudder cmd
Initial_Acceleration_Proportional_Gain	Acceleration_Gains	Proportional gain in integrator plus proportional gain loop

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Aileron control loop limiters and filters		
Roll_Error_Limit	function	Limiter for roll error
Aileron_Command_Limit	function	Limit on command signal to aileron
Roll_Command_Filter	function	Filter applied to input roll command
Rudder control loop limiters, filters, and operations		
Rudder_Command_Limit	function	Limit on command signal to rudder
Yaw_Rate_Filter	function	Filter applied to measured yaw rate
Acceleration_Filter	function	Filter applied to measured acceleration feedback
Sin	function	Sin function applied to measured roll attitude
Aileron control loop gain and updater functions		
"-"	function	Subtracts Roll_Attitudes from Roll_Commands returning Roll_Error
"*"	function	Multiplies Roll_Commands by Roll_Command_Gains for input to Aileron integrator
"*"	function	Multiplies Feedback_Rates for measured roll rate by Feedback_Rate_Gains for Fin_Deflections
Rudder control loop gain and updater functions		
"*"	function	Multiplies Missile_Accelerations by Acceleration_Gains returns Fin_Deflections for proportional loop of integral plus proportional gain
"*"	function	Multiplies Feedback_Rates by Rudder_Cmd_Roll_Rate_Gains returns Feedback_Rates
"*"	function	Multiplies Gravitational Accelerations by Trig_Value returns Gravitational_Accelerations
"/"	function	Divides Gravitational Accelerations by Velocities returns Feedback_Rates

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Description
Aileron_Rudder_Commands	Record containing components for Aileron and Rudder Command

3.6.6.2.9.3.3 LOCAL ENTITIES

Data structures:

Stores values for gains and limits.

Packages:

Instantiates Integral plus proportional gain package for aileron roll command and for filtered lateral directional acceleration. Also instantiates Tustin integrator to implement each of the integral plus proportional gain packages.

3.6.6.2.9.3.4 INTERRUPTS

None.

3.6.6.2.9.3.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types, Basic_Data_Types, Autopilot;
use Autopilot_Data_Types, Basic_Data_Types, Basic_Data_Types.Trig;
procedure USER is
   package A_D_T renames Autopilot_Data_Types;

   package B_D_T renames Basic_Data_Types;
   type Acceleration_Filter_Coefficients is new
                                   A_D_T.Acceleration_Feedbacks_FPS2;


   Previous_Roll_Command           : A_D_T.Roll_Commands_Radians    := 1.0;
   Measured_Acceleration_Feedback  : A_D_T.Acceleration_Feedbacks_FPS2
                                                                    := 1.0;
   Yaw_Rate_Feedback               : A_D_T.Feedback_Rates_Radians   := 1.0;
   Aileron_Proportional_Gain       : A_D_T.Gain_In_Radians          := 1.0;
   Aileron_Integrator_Gain         : A_D_T.Gain_In_Radians          := 0.1;
   Aileron_Integrator_Limit        : A_D_T.Fin_Deflections_Radians  := 5.0;
   Aileron_Command_Roll_Rate_Gain  : A_D_T.RPS_To_Radians_Gain      := 0.1;
   Aileron_Command_Yaw_Rate_Gain   : A_D_T.RPS_To_Radians_Gain      := 0.1;
   Rudder_Integrator_Gain          : A_D_T.FPS2_To_Radians_Gain     := 0.1;
   Rudder_Integrator_Limit         : A_D_T.Fin_Deflections_Radians  := 5.0;
   Rudder_Proportional_Gain        : A_D_T.FPS2_To_Radians_Gain     := 0.1;
```

```
Yaw_Rate_Gain                   : A_D_T.RPS_To_Radians_Gain      := 0.1;
Rudder_Roll_Rate_Gain           : A_D_T.Feedback_Rates_Radians   := 0.1;
Acceleration_Gain               : A_D_T.FPS2_To_Radians_Gain     := 0.1;

package Roll_Error_Limiter is new
           Signal_Processing.Absolute_Limiter_With_Flag
              (Signal_Type            => A_D_T.Roll_Commands_Radians,
               Initial_Absolute_Limit => A_D_T.Roll_Commands_Radians (5.0));

function Roll_Error_Limit (Roll_Command: A_D_T.Roll_Commands_Radians)
   return A_D_T.Roll_Commands_Radians
   renames Roll_Error_Limiter.Limit;

package Aileron_Command_Limiter is new
           Signal_Processing.Absolute_Limiter_With_Flag
             (Signal_Type            => A_D_T.Fin_Deflections_Radians,
              Initial_Absolute_Limit => A_D_T.Fin_Deflections_Radians (5.0));

function Aileron_Command_Limit
              (Fin_Deflection: A_D_T.Fin_Deflections_Radians)
   return A_D_T.Fin_Deflections_Radians
   renames Aileron_Command_Limiter.Limit;

package Roll_Commands_Filter is new
           Signal_Processing.Tustin_Lag_Filter
              (Signal_Type                => A_D_T.Roll_Commands_Radians,
               Coefficient_Type           => A_D_T.Roll_Commands_Radians,
               Initial_Previous_Input_Signal
                                          => Previous_Roll_Command,
               Initial_Coefficient_1      => A_D_T.Roll_Commands_Radians (0.9),
               Initial_Coefficient_2      => A_D_T.Roll_Commands_Radians (0.1));

function Roll_Command_Filter
                    (Roll_Commands : A_D_T.Roll_Commands_Radians)
   return A_D_T.Roll_Commands_Radians
   renames Roll_Commands_Filter.Filter;

package Rudder_Command_Limiter is new
           Signal_Processing.Absolute_Limiter_With_Flag
             (Signal_Type            => A_D_T.Fin_Deflections_Radians,
              Initial_Absolute_Limit => A_D_T.Fin_Deflections_Radians (5.0));

function Rudder_Command_Limit
              (Fin_Deflection: A_D_T.Fin_Deflections_Radians)
   return A_D_T.Fin_Deflections_Radians
   renames Rudder_Command_Limiter.Limit;

package Yaw_Rates_Filter is new
           Signal_Processing.Tustin_Lag_Filter
              (Signal_Type                => A_D_T.Feedback_Rates_Radians,
               Coefficient_Type           => A_D_T.Feedback_Rates_Radians,
               Initial_Previous_Input_Signal
                                          => Yaw_Rate_Feedback,
               Initial_Coefficient_1      => A_D_T.Feedback_Rates_Radians (0.9),
               Initial_Coefficient_2      => A_D_T.Feedback_Rates_Radians (0.1));

function Yaw_Rate_Filter
```

```
                    (Yaw_Rate : A_D_T.Feedback_Rates_Radians)
      return A_D_T.Feedback_Rates_Radians
      renames Yaw_Rates_Filter.Filter;

   package Accelerations_Filter is new
            Signal_Processing.Tustin_Lag_Filter
               (Signal_Type               => A_D_T.Acceleration_Feedbacks_FPS2,
                Coefficient_Type          => A_D_T.Acceleration_Feedbacks_FPS2,
                Initial_Previous_Input_Signal
                                          => Measured_Acceleration_Feedback,
                Initial_Coefficient_1     => A_D_T.Acceleration_Feedbacks_FPS2 (0.9),
                Initial_Coefficient_2     => A_D_T.Acceleration_Feedbacks_FPS2 (0.1));

   function Acceleration_Filter
             (Lateral_Accelerations: A_D_T.Acceleration_Feedbacks_FPS2)
      return A_D_T.Acceleration_Feedbacks_FPS2
      renames Accelerations_Filter.Filter;

   package Lat_Dir_Autopilot is new
            Autopilot.Lateral_Directional_Autopilot
      (Roll_Commands                 => A_D_T.Roll_Commands_Radians,
       Roll_Attitudes                => A_D_T.Missile_Attitudes_Radians,
       Roll_Command_Gains            => A_D_T.Gain_In_Radians,
       Missile_Accelerations         => A_D_T.Acceleration_Feedbacks_FPS2,
       Acceleration_Gains            => A_D_T.FPS2_To_Radians_Gain,
       Rudder_Cmd_Roll_Rate_Gains    => A_D_T.Feedback_Rates_Radians,
       Gravitational_Accelerations   => BDT.GEES,
       Velocities                    => BDT.Feet_Per_Second,
       Trig_Value                    => BDT.Trig.Sin_Cos_Ratio,
       Fin_Deflections               => A_D_T.Fin_Deflections_Radians,
       Feedback_Rates                => A_D_T.Feedback_Rates_Radians,
       Feedback_Rate_Gains           => A_D_T.RPS_To_Radians_Gain,

       Initial_Aileron_Integrator_Gain
                                     => Aileron_Integrator_Gain,
       Initial_Aileron_Integrator_Limit
                                     => Aileron_Integrator_Limit,
       Initial_Roll_Command_Proportional_Gain
                                     => Aileron_Proportional_Gain,
       Initial_Roll_Rate_Gain_For_Aileron
                                     => Aileron_Command_Roll_Rate_Gain,
       Initial_Yaw_Rate_Gain_For_Aileron
                                     => Aileron_Command_Roll_Rate_Gain,
       Initial_Rudder_Integrator_Gain
                                     => Rudder_Integrator_Gain,
       Initial_Rudder_Integrator_Limit
                                     => Rudder_Integrator_Limit,
       Initial_Yaw_Rate_Gain_For_Rudder
                                     => Yaw_Rate_Gain,
       Initial_Roll_Rate_Gain_For_Rudder
                                     => Rudder_Roll_Rate_Gain,
       Initial_Acceleration_Proportional_Gain
                                     => Rudder_Proportional_Gain
           -- all overloaded operators default
                                                                         );

   Aileron_Command      : A_D_T.Fin_Deflections_Radians;
   Rudder_Command       : A_D_T.Fin_Deflections_Radians;
```

```
   Grav_Acceleration    : A_D_T.Acceleration_Feedbacks_FPS2;
   Roll_Command         : A_D_T.Roll_Commands_Radians;
   Roll_Attitude        : A_D_T.Missile_Attitudes_Radians;
   Roll_Rate            : A_D_T.Feedback_Rates_Radians;
   Yaw_Rate             : A_D_T.Feedback_Rates_Radians;
   Velocity             : B_D_T.Feet_Per_Second;
   Lat_Acceleration     : A_D_T.Acceleration_Feedbacks_FPS2;

   Aileron_Rudder_Commands : Lat_Dir_Autopilot.Aileron_Rudder_Commands;

begin

   -- obtain Normal_Acceleration_Command
   -- obtain Acceleration_ and Pitch_Rate_Feedbacks

   Lat_Dir_Autopilot.Initialize_Lateral_Directional_Autopilot
        (Initial_Aileron_Command        => Aileron_Command,
         Initial_Rudder_Command         => Rudder_Command,
         Gravitational_Acceleration     => Grav_Acceleration,
         Roll_Command                   => Roll_Command,
         Roll_Attitude                  => Roll_Attitude,
         Roll_Rate                      => Roll_Rate,
         Yaw_Rate                       => Yaw_Rate,
         Missile_Velocity               => Velocity,
         Lateral_Acceleration           => Lat_Acceleration);

   Aileron_Rudder_Commands := Lat_Dir_Autopilot.Compute_Aileron_Rudder_Commands
      (Roll_Command                   => Roll_Command,
                Roll_Attitude                   => Roll_Attitude,
        Roll_Rate                     => Roll_Rate,
         Yaw_Rate                       => Yaw_Rate,
         Lateral_Acceleration           => Lat_Acceleration,
         Missile_Velocity               => Velocity,
         Gravitational_Acceleration     => Grav_Acceleration);

end USER;
```

3.6.6.2.9.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.6.2.9.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Initialize_Lateral_Directional_Autopilot	Procedure	Sets initial states of lateral directional autopilot by initializing integrator states of roll command and lateral acceleration
Compute_Aileron_Rudder_Commands	Function	Accepts roll command, and measured values for roll rate, roll attitude, yaw rate, and lateral acceleration and uses integrator states to calculate fin deflections for rudder and aileron
Update_Aileron_Integrator_Gain	Procedure	Changes value stored for gain on roll command to aileron integrator
Update_Aileron_Integrator_Limit	Procedure	Changes value stored for limit to integrator output
Update_Roll_Command_Proportional_Gain	Procedure	Changes value stored for proportional gain in integral plus proportional gain loop
Update_Roll_Rate_Gain_For_Aileron	Procedure	Changes value stored for gain to measured roll rate for aileron control
Update_Rudder_Integrator_Gain	Procedure	Changes value stored for gain to lateral acceleration value in integrator loop
Update_Rudder_Integrator_Limit	Procedure	Changes value stored for limit to rudder integrator output
Update_Feedback_Rate_Gain_For_Rudder	Procedure	Changes value stored for gain to measure yaw rate in rudder control loop
Update_Role_Rate_Gain_For_Rudder	Procedure	Changes value stored for gain to measured roll rate for rudder control loop
Update_Acceleration_Proportional_Gain	Procedure	Changes value stored for gain in proportional loop

3.6.6.2.9.3.8 PART DESIGN

None.

```
package Autopilot is
pragma PAGE;
generic
   type Input_Signals        is digits <>;
   type Gains                is digits <>;
   type Integrated_Signals   is digits <>;
   Initial_Proportional_Gain: in Gains;
   with function "*" (Left   : Input_Signals;
                      Right  : Gains) return Integrated_Signals is <>;
   with function Tustin_Integrate (Signal: in Input_Signals)
                      return Integrated_Signals is <>;
package Integral_Plus_Proportional_Gain is

   function Integrate (Signal: Input_Signals) return Integrated_Signals;

   procedure Update_Proportional_Gain (New_Proportional_Gain : in Gains);

end Integral_Plus_Proprotional_Gain;

                                                          pragma PAGE;
--   --Pitch Autopilot Package

    generic

       type Normal_Acceleration_Commands is digits <>;
       type Accelerations                is digits <>;
       type Acceleration_Command_Gains   is digits <>;
       type Acceleration_Gains           is digits <>;
       type Pitch_Rates                  is digits <>;
       type Pitch_Rate_Gains             is digits <>;
       type Fin_Deflections              is digits <>;

--    --Initial values for pitch control loop

       Initial_Integrator_Gain   : in Acceleration_Command_Gains;
       Initial_Integrator_Limit  : in Fin_Deflections;
       Initial_Acceleration_Gain : in Acceleration_Gains;
       Initial_Pitch_Rate_Gain   : in Pitch_Rate_Gains;
       Initial_Proportional_Gain : in Acceleration_Command_Gains;

--     --Pitch control loop limiter

       with function Limit (Elevator_Command : Fin_Deflections)
            return Fin_Deflections is <>;

--     --Pitch control loop filters

       with function Acceleration_Filter  (Normal_Acceleration : Accelerations)
            return Accelerations is <>;

       with function Pitch_Rate_Filter (Pitch_Rate : in Pitch_Rates)
            return Pitch_Rates is <>;

--     --Gain and updater operations

       with function "*" (Left  : Normal_Acceleration_Commands;
                          Right : Acceleration_Command_Gains)
```

```
            return Fin_Deflections is <>;

        with function "*" (Left  : Accelerations;
                           Right : Acceleration_Gains)
            return Fin_Deflections is <>;

        with function "+" (Left  : Normal_Acceleration_Commands;
                           Right : Accelerations)
            return Normal_Acceleration_Commands is <>;

        with function "-" (Left  : Normal_Acceleration_Commands;
                           Right : Accelerations)
            return Normal_Acceleration_Commands is <>;

        with function "*" (Left  : Pitch_Rates;
                           Right : Pitch_Rate_Gains)
            return Fin_Deflections is <>;

    package Pitch_Autopilot is

        procedure Initialize_Pitch_Autopilot
                (Normal_Acceleration_Command   : in Normal_Acceleration_Commands;
                 Measured_Normal_Acceleration  : in Accelerations;
                 Measured_Pitch_Rate           : in Pitch_Rates;
                 Initial_Elevator_Command      : in Fin_Deflections);

        function Compute_Elevator_Command
                (Normal_Acceleration_Command  : in Normal_Acceleration_Commands;
                 Measured_Normal_Acceleration : in Accelerations;
                 Measured_Pitch_Rate          : in Pitch_Rates)
           return Fin_Deflections;

        procedure Update_Pitch_Rate_Gain (New_Gain: in Pitch_Rate_Gains);

        procedure Update_Acceleration_Gain (New_Gain: in Acceleration_Gains);

        procedure Update_Integrator_Gain
                    (New_Gain: in Acceleration_Command_Gains);

        procedure Update_Integrator_Limit (New_Limit: in Fin_Deflections);

        procedure Update_Proportional_Gain
                    (New_Proportional_Gain : in Acceleration_Command_Gains);

    end Pitch_Autopilot;

                                                            pragma PAGE;
-- --Lateral/Directional Autopilot Package

   generic

--       --types for Aileron Loop

      type Roll_Commands            is digits <>;
      type Roll_Attitudes           is digits <>;
      type Roll_Command_Gains       is digits <>;
```

```
--      --types for Rudder Loop

        type Missile_Accelerations      is digits <>;
        type Acceleration_Gains         is digits <>;
        type Rudder_Cmd_Roll_Rate_Gains is digits <>;
        type Gravitational_Accelerations is digits <>;
        type Velocities                 is digits <>;
        type Trig_Value                 is digits <>;

--      --types for both loops

        type Fin_Deflections            is digits <>;
        type Feedback_Rates             is digits <>;
        type Feedback_Rate_Gains        is digits <>;

--      --Initial values for aeliron control loop

        Initial_Aileron_Integrator_Gain         : in Roll_Command_Gains;
        Initial_Aileron_Integrator_Limit        : in Fin_Deflections;
        Initial_Roll_Command_Proportional_Gain  : in Roll_Command_Gains;
        Initial_Roll_Rate_Gain_For_Aileron      : in Feedback_Rate_Gains;
        Initial_Yaw_Rate_Gain_For_Aileron       : in Feedback_Rate_Gains;

--      --Initial values for rudder control loop

        Initial_Rudder_Integrator_Gain          : in Acceleration_Gains;
        Initial_Rudder_Integrator_Limit         : in Fin_Deflections;
        Initial_Yaw_Rate_Gain_For_Rudder        : in Feedback_Rate_Gains;
        Initial_Roll_Rate_Gain_For_Rudder       : in Rudder_Cmd_Roll_Rate_Gains;
        Initial_Acceleration_Proportional_Gain  : in Acceleration_Gains;

--      --Aileron control loop limiters and filter

        with function Roll_Error_Limit (Roll_Command : Roll_Commands)
              return Roll_Commands is <>;

        with function Aileron_Command_Limit (Fin_Deflection: Fin_Deflections)
              return Fin_Deflections is <>;

        with function Roll_Command_Filter (Roll_Command: Roll_Commands)
              return Roll_Commands is <>;

--      --Rudder control loop limiters, filters, and trig function

        with function Rudder_Command_Limit (Fin_Deflection: Fin_Deflections)
              return Fin_Deflections is <>;

        with function Yaw_Rate_Filter (Yaw_Rate: Feedback_Rates)
              return Feedback_Rates is <>;

        with function Acceleration_Filter (Lateral_Acceleration: Missile_Accelerations)
              return Missile_Accelerations is <>;

        with function Sin (Angle: Roll_Attitudes) return Trig_Value is <>;

--      --Aileron control loop gain and updater functions
```

```
      with function "-" (Left  : Roll_Commands;
                         Right : Roll_Attitudes)
           return Roll_Commands is <>;

      with function "*" (Left  : Roll_Commands;
                         Right : Roll_Command_Gains)
           return Fin_Deflections is <>;

      with function "*" (Left  : Feedback_Rates;
                         Right : Feedback_Rate_Gains)
           return Fin_Deflections is <>;

--    --Rudder control loop gain and updater functions

      with function "*" (Left  : Missile_Accelerations;
                         Right : Acceleration_Gains)
           return Fin_Deflections is <>;

      with function "*" (Left  : Feedback_Rates;
                         Right : Rudder_Cmd_Roll_Rate_Gains)
           return Feedback_Rates is <>;

      with function "*" (Left  : Gravitational_Accelerations;
                         Right : in Trig_Value)
           return Gravitational_Accelerations is <>;

      with function "/" (Left  : Gravitational_Accelerations;
                         Right : Velocities)
           return Feedback_Rates is <>;

   package Lateral_Directional_Autopilot is

      type Aileron_Rudder_Commands is record
            Aileron_Command : Fin_Deflections;
            Rudder_Command  : Fin_Deflections;
      end record;

      procedure Initialize_Lateral_Directional_Autopilot
                   (Initial_Aileron_Command     : in Fin_Deflections;
                    Initial_Rudder_Command      : in Fin_Deflections;
                    Gravitational_Acceleration : in Gravitational_Accelerations;
                    Roll_Command                : in Roll_Commands;
                    Roll_Attitude               : in Roll_Attitudes;
                    Roll_Rate                   : in Feedback_Rates;
                    Yaw_Rate                    : in Feedback_Rates;
                    Missile_Velocity            : in Velocities;
                    Lateral_Acceleration        : in Missile_Accelerations);

      function Compute_Aileron_Rudder_Commands
                  (Roll_Command               : in Roll_Commands;
                   Roll_Attitude              : in Roll_Attitudes;
                   Roll_Rate                  : in Feedback_Rates;
                   Yaw_Rate                   : in Feedback_Rates;
                   Lateral_Acceleration       : in Missile_Accelerations;
                   Missile_Velocity           : in Velocities;
                   Gravitational_Acceleration : in Gravitational_Accelerations)
         return Aileron_Rudder_Commands;
```

```
      procedure Update_Aileron_Integrator_Gain
                  (New_Gain : in Roll_Command_Gains);

      procedure Update_Aileron_Integrator_Limit
                  (New_Limit : in Fin_Deflections);

      procedure Update_Roll_Command_Proportional_Gain
                  (New_Gain : in Roll_Command_Gains);

      procedure Update_Roll_Rate_Gain_For_Aileron
                  (New_Gain : in Feedback_Rate_Gains);

      procedure Update_Yaw_Rate_Gain_For_Aileron
                  (New_Gain : in Feedback_Rate_Gains);

      procedure Update_Rudder_Integrator_Gain
                  (New_Gain : in Acceleration_Gains);

      procedure Update_Rudder_Integrator_Limit
                  (New_Limit : in Fin_Deflections);

      procedure Update_Feedback_Rate_Gain_For_Rudder
                  (New_Gain : in Feedback_Rate_Gains);

      procedure Update_Roll_Rate_Gain_For_Rudder
                  (New_Gain : in Rudder_Cmd_Roll_Rate_Gains);

      procedure Update_Acceleration_Proportional_Gain
                  (New_Gain : in Acceleration_Gains);

   end Lateral_Directional_Autopilot;

end Autopilot;
```

(This page left intentionally blank.)

3.6.7 NON-GUIDANCE CONTROL

(This page left intentionally blank.)

3.6.7.1 AIR_DATA (PACKAGE SPECIFICATION) TLCSC (CATALOG #P309-0)

This TLCSC contains parts which can be used to monitor air conditions.

3.6.7.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Compute_Outside_Air_Temperature	R228
Compute_Pressure_Ratio	R229
Compute_Mach	R230
Compute_Dynamic_Pressure	R231
Compute_Speed_Of_Sound	R232
Barometric_Altitude_Integration	R233

3.6.7.1.2 INPUT/OUTPUT

None.

3.6.7.1.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.7.1.4 LOCAL ENTITIES

None.

3.6.7.1.5 INTERRUPTS

None.

3.6.7.1.6 TIMING AND SEQUENCING

None.

3.6.7.1.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.7.1.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Compute_Outside_Air_Temperature	generic function	Computes air temperature outside a missile
Compute_Pressure_Ratio	generic function	Computes pressure ratio from a collection of pressures
Compute_Mach	generic function	Computes missile mach (velocity) given pressure ratio
Compute_Dynamic_Pressure	generic function	Computes dynamic pressure from mach # & free stream static pressre
Compute_Speed_Of_Sound	generic function	Computes speed of sound given the temperature of the air
Barometric_Altitude_Integration	generic package	Computes barometric altitude by integration of the atmospheric equation of state

3.6.7.1.9 PART DESIGN

3.6.7.1.9.1 AIR_DATA.COMPUTE_OUTSIDE_AIR_TEMPERATURE (FUNCTION SPECIFICATION) (CATALOG #P310-0)

This unit is a generic function which computes air temperature outside of a missile.

3.6.7.1.9.1.1 REQUIREMENTS ALLOCATION

This parts meets CAMP Requirement R228

3.6.7.1.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Temperatures	floating point type	Describes air temperatures
Mach_Numbers	floating point type	Describes air speed as a ratio of the speed of sound
Real	floating point type	General floating point type

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Recovery_Factor	Real	N/A	Constant for computing Air Temp

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies a Real by a Mach Number, yielding a Mach Number
"/"	function	Divides a Temperature by a Mach Number, yielding a Temperature

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Total_ Temperature	Temperatures	in	Air temperature measured by the air data instruments
Mach	Mach_Numbers	in	Missile airspeed as a fraction of the speed of sound
<returned value>	Temperatures	out	Temperature of the air outside of the missile

3.6.7.1.9.1.3 INTERRUPTS

None.

3.6.7.1.9.1.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Air_Data, Basic_Data_Types;
procedure Test is

   package BDT renames Basic_Data_Types;
   subtype Mach_Numbers is FLOAT;
   subtype Real         is FLOAT;

   function Outside_Air_Temp is new Air_Data.Compute_Outside_Air_Temperature
                (Temperatures    => BDT.Degrees_Kelvin,
                 Mach_Numbers    => Mach_Numbers,
                 Real            => Real,
                 Recovery_Factor => 0.7);

   Total_Temp : BDT.Degrees_Kelvin;
   Mach       : Mach_Numbers;

begin
   Outside_Temp := Outside_Air_Temp (Total_Temp, Mach);
end Test;
```

3.6.7.1.9.1.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.7.1.9.1.6 DECOMPOSITION

None.

3.6.7.1.9.2 AIR_DATA.COMPUTE_PRESSURE_RATIO (FUNCTION SPECIFICATION) (CATALOG #P311-0)

This unit is a generic function which computes pressure ratio from measured static pressure, measured impact pressure, and free stream static pressure.

3.6.7.1.9.2.1 REQUIREMENTS ALLOCATION

This parts meets CAMP requirement R229

3.6.7.1.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Pressures	floating point type	Describes pressure (i.e. weight per unit of area)
Ratios	floating point type	A unitless floating point type describing ratio of one pressure to another

Subprograms:

The following table de. `bes the generic formal subroutines required by this part:

Name	Type	Description
"/"	function	Divides a Pressure by a Pressure, yielding a ratio

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Measured_Static_Pressure	Pressures	in	Static pressure measured by the air data system
Impact_Pressure	Pressures	in	Measured difference between totl pressure and static pressure
Free_Stream_Static_Pressure	Pressures	in	Measured static pressure which has been corrected for errors
<returned value>	Ratios	out	Unitless quantity computed from static and impact pressure

3.6.7.1.9.2.3 INTERRUPTS

None.

3.6.7.1.9.2.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Air_Data, Basic_Data_Types;
```

```
procedure Test is

   package BDT renames Basic_Data_Types;
   subtype Ratios is FLOAT;

   function Pressure_Ratio is new Air_Data.Compute_Pressure_Ratio
               (Pressures => BDT.Kgs_Per_Meter_Squared,
                Ratios    => Ratios);

   Measured_Static_Pressure : BDT.Kgs_Per_Meter_Squared;
   Impact_Pressure          : BDT.Kgs_Per_Meter_Squared;
   Free_Static_Pressure     : BDT.Kgs_Per_Meter_Squared;
   Ratio                    : Ratios;

begin
   Ratio := Pressure_Ratio (Measured_Static_Pressure,
                            Impact_Pressure,
                            Free_Static_Pressure);
end Test;
```

3.6.7.1.9.2.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.7.1.9.2.6 DECOMPOSITION

None.

3.6.7.1.9.3 AIR_DATA.COMPUTE_MACH (FUNCTION SPECIFICATION) (CATALOG #P312-0)

This unit is a generic function which computes missile mach given pressure ratio.

3.6.7.1.9.3.1 REQUIREMENTS ALLOCATION

This parts meets CAMP requirement R230.

3.6.7.1.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Mach_Numbers	floating point type	Describes air speed as a ratio of the speed of sound
Ratios	floating point type	A unitless floating point type descri-ing ratio of one pressure to another

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
C0	Ratios		First curve fit parameter
C1	Ratios		Second curve fit parameter
C2	Ratios		Third curve fit parameter

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Sqrt	function	Computes the square root of Ratio, yielding a Mach Number

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Pressure_Ratio	Ratios	in	Unitless quantity computed from static and impact pressures
<returned value>	Mach_Numbers	out	Missile airspeed as a fraction of the speed of sound

3.6.7.1.9.3.3 INTERRUPTS

None.

3.6.7.1.9.3.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Air_Data;
procedure Test is

   subtype Ratios is FLOAT;
   subtype Mach_Numbers is FLOAT;

   function Computed_Mach is new Air_Data.Compute_Mach
               (Ratios       => Ratios,
                Mach_Numbers => Mach_Numbers,
                C0           => 0.1,
                C1           => 0.2,
                C2           => 0.3);

   Pressure_Ratio : Ratios;
   Mach           : Mach_Numbers;

begin
   Mach := Computed_Mach (Pressure_Ratio);
end Test;
```

3.6.7.1.9.3.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.7.1.9.3.6 DECOMPOSITION

None.

3.6.7.1.9.4 AIR_DATA.COMPUTE_DYNAMIC_PRESSURE (FUNCTION SPECIFICATION) (CATALOG #P313-0)

This unit is a generic function which computes dynamic pressure from missile mach number and free stream static pressure.

3.6.7.1.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP Requirement R231.

3.6.7.1.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Pressures	floating point type	Describes pressure (i.e. weight per unit of area)
Mach_Numbers	floating point type	Describes air speed as a ratio of the speed of sound

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies a Pressure by a Mach Number, yielding a Pressure

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Free_Stream_ Static_Pressure	Pressures	in	Measured static pressure which has been corrected for errors
Mach	Mach_Numbers	in	Missile airspeed as a fraction of the speed of sound
<returned value>	Pressures	out	Missile dynamic pressure

3.6.7.1.9.4.3 INTERRUPTS

None.

3.6.7.1.9.4.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Air_Data, Basic_Data_Types;
procedure Test is

   package BDT renames Basic_Data_Types;
   subtype Mach_Numbers is FLOAT;

   function Dynamic_Pressure is new Air_Data.Compute_Dynamic_Pressure
               (Pressures    => BDT.Kgs_Per_Meter_Squared,
                Mach_Numbers => Mach_Numbers);
```

```
   Free_Stream_Static_Pressure : BDT.Kgs_Per_Meter_Squared;
   Pressure                    : BDT.Kgs_Per_Meter_Squared;
   Mach                        : Mach_Numbers;

begin
   Pressure := Dynamic_Pressure (Free_Stream_Static_Pressure,
                                 Mach);
end Test;
```

3.6.7.1.9.4.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.7.1.9.4.6 DECOMPOSITION

None.

3.6.7.1.9.5 AIR_DATA.COMPUTE_SPEED_OF_SOUND (FUNCTION SPECIFICATION) (CATALOG #P314-0)

This unit is a generic funciton which computes the speed of sound given the temperature of the air.

3.6.7.1.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R232.

3.6.7.1.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Temperatures	floating point type	Describes air temperatures
Velocities	floating point type	Describes air speed

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Speed_Of_ Sound_ Constant	Velocities	N/A	Standard speed of sound at sea level

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies a Velocity by a Temperature, yielding a Velocity
Sqrt	function	Computes the square root of a Temperature, yielding a Temperature

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Air_Temperature	Temperatures	in	Temperature of the air
<returned value>	Velocities	out	Speed of sound in air

3.6.7.1.9.5.3 INTERRUPTS

None.

3.6.7.1.9.5.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Air_Data, Basic_Data_Types, WGS72_Metric_Data;
procedure Test is

   package BDT renames Basic_Data_Types;

   function Speed_Of_Sound is new Air_Data.Compute_Speed_Of_Sound
               (Temperatures          => BDT.Degrees_Kelvin,
                Velocities            => BDT.Meters_Per_Second,
                Speed_Of_Sound_Constant => 630.0);

   Air_Temperature : Temperatures;
   Speed           : Velocities;
```

```
begin
   Speed := Speed_Of_Sound (Air_Temperature => Air_Temperature);
end Test;
```

3.6.7.1.9.5.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.7.1.9.5.6 DECOMPOSITION

None.

3.6.7.1.9.6 AIR_DATA.COMPUTE_BAROMETRIC_ALTITUDE (PACKAGE SPECIFICATION) (CATALOG #P315-0)

This unit is a generic function which computes barometric altitude by integration of the atmospheric equation of state.

3.6.7.1.9.6.1 REQUIREMENTS ALLOCATION

This parts meets CAMP requirement R233.

3.6.7.1.9.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Temperatures	floating point type	Describes air temperatures
Pressures	floating point type	Describes pressure (i.e. weight per unit of area)
Distances	floating point type	Describes translational distances (e.g., Feet, Meters)
Molar_Gas_Constants	floating point type	Describes the type of the Gas Constant needed

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Gas_Constant	Molar_Gas_Constants	N/A	Constant which describes a standard gas constant
Maximum_Pressure_Change	Pressures	N/A	Maximum reasonable change expected in free stream static pressure between two measurement
Intial_Baro_Altitude	Distances	N/A	Barometric Altitude at the start of integration

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies the Gas Constant by a Pressure yielding a Distance
"/"	function	Divides a Temperature by a Pressure, yielding a Pressure

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Outside_Air_Temperature	Temperatures	in	Temperature of the air outside the missile
Free_Stream_Static_Pressure	Pressures	in	Measured static pressure corrected for errors
<returned value>	Distances	out	Altitude in feet based on the barometric pressure of the atmosphere

3.6.7.1.9.6.3 LOCAL ENTITIES

None.

3.6.7.1.9.6.4 INTERRUPTS

None.

3.6.7.1.9.6.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Air_Data, Basic_Data_Types;
procedure Test is

   package BDT renames Basic_Data_Types;

   function Baro_Altitude is new Air_Data.Barometric_Altitude_Integration
              (Temperatures            => BDT.Degrees_Kelvin,
               Pressures               => BDT.Kgs_Per_Meter_Squared,
               Distances               => BDT.Meters,
               Molar_Gas_Constants     => Gas_Constants,
               Gas_Constant            => 0.04563,
               Maximum_Pressure_Change => 0.9765,
               Initial_Baro_Altitude   => 1500.0);

   Air_Temperature : BDT.Degrees_Kelvin;
   Static_Pressure : BDT.Kgs_Per_Meter_Squared;
   Altitude        : BDT.Meters;
begin
   Altitude := Baro_Altitude (Outside_Air_Temperature => Air_Temperature,
                              Free_Stream_Static_Pressure => Static_Pressure);
end Test;
```

3.6.7.1.9.6.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.7.1.9.6.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Compute_Barometric_ Altitude	generic package	Computes barometric altitude by integration of the atmospheric equation of state

3.6.7.1.9.6.8 PART DESIGN

None.

(This page left intentionally blank.)

```
package Air_Data_Parts is
                                                            pragma PAGE;
   generic
      type Temperatures is digits <>;
      type Mach_Numbers is digits <>;
      type Real         is digits <>;

      Recovery_Factor : in Real;

      with function "*" (Left  : Real;
                         Right : Mach_Numbers)
                        return Mach_Numbers is <>;
      with function "/" (Left  : Temperatures;
                         Right : Mach_Numbers)
                        return Temperatures is <>;

   function Compute_Outside_Air_Temperature
              (Total_Temperature : Temperatures;
               Mach              : Mach_Numbers)
              return Temperatures;
                                                            pragma PAGE;
   generic
      type Pressures is digits <>;
      type Ratios    is digits <>;

      with function "/" (Left  : Pressures;
                         Right : Pressures)
                        return Ratios is <>;

   function Compute_Pressure_Ratio
              (Measured_Static_Pressure    : Pressures;
               Impact_Pressure             : Pressures;
               Free_Stream_Static_Pressure : Pressures)
              return Ratios;
                                                            pragma PAGE;
   generic
      type Ratios              is digits <>;
      type Mach_Numbers        is digits <>;

      C0 : in Ratios;
      C1 : in Ratios;
      C2 : in Ratios;

      with function Sqrt (Source : Ratios)
                         return Mach_Numbers is <>;

   function Compute_Mach
              (Pressure_Ratio : Ratios)
              return Mach_Numbers;
                                                            pragma PAGE;
   generic
      type Pressures    is digits <>;
      type Mach_Numbers is digits <>;

      with function "*" (Left  : Pressures;
                         Right : Mach_Numbers)
                        return Pressures is <>;
```

```
   function Compute_Dynamic_Pressure
               (Free_Stream_Static_Pressure : Pressures;
                Mach                        : Mach_Numbers)
               return Pressures;
                                                                    pragma PAGE;
   generic
      type Temperatures is digits <>;
      type Velocities   is digits <>;

      Speed_Of_Sound_Constant : in Velocities;

      with function "*" (Left  : Velocities;
                         Right : Temperatures)
                        return Velocities is <>;
      with function Sqrt (Source : Temperatures)
                         return Temperatures is <>;

   function Compute_Speed_Of_Sound
               (Air_Temperature : Temperatures)
               return Velocities;
                                                                    pragma PAGE;
   generic
      type Temperatures       is digits <>;
      type Pressures          is digits <>;
      type Distances          is digits <>;
      type Molar_Gas_Constants is digits <>;

      Gas_Constant                : in Molar_Gas_Constants;
      Maximum_Pressure_Change     : in Pressures;
      Initial_Free_Stream_Pressure : in Pressures;
      Initial_Temperature         : in Temperatures;
      Initial_Baro_Altitude       : in Distances;

      with function "*" (Left  : Molar_Gas_Constants;
                         Right : Pressures)
                        return Distances is <>;
      with function "/" (Left  : Temperatures;
                         Right : Pressures)
                        return Pressures is <>;

   package Barometric_Altitude_Integration is

      function Compute_Barometric_Altitude
                  (Outside_Air_Temperature     : Temperatures;
                   Free_Stream_Static_Pressure : Pressures)
                  return Distances;

   end Barometric_Altitude_Integration;

end Air_Data_Parts;
```

3.6.7.2 FUEL_CONTROL_PARTS TLCSC P672 (CATALOG #P1095-0)

This TLCSC contains parts which can be used to maintain control over fuel in missile applications.

3.6.7.2.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Throttle_Command_Manager	R234

3.6.7.2.2 INPUT/OUTPUT

None.

3.6.7.2.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.7.2.4 LOCAL ENTITIES

None.

3.6.7.2.5 INTERRUPTS

None.

3.6.7.2.6 TIMING AND SEQUENCING

None.

3.6.7.2.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.7.2.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Throttle_Command_Manager	generic package	Manages the throttle command missile

3.6.7.2.9 PART DESIGN

3.6.7.2.9.1 THROTTLE_COMMAND_MANAGER

This LLCSC is a generic package which manages the throttle command.

3.6.7.2.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP Requirement R234.

3.6.7.2.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types: The following table describes the generic formal types required by this part:

Name	Type	Description
Mach_Numbers	floating point type	Represents missile speed as a percentage of the speed of sound
Mach_Number_Gains	floating point type	Represents a gain which converts from Mach Number to Throttle Command
Throttle_Commands	floating point type	Represents a command to open/close the throttle

Data objects: The following table describes the generic formal objects required by this part:

Name	Type	Mode	Description
Initial_Mach_Command	Mach_Numbers	in	Mach Number of missile at startup
Initial_Mach_Error_Limit	Mach_Numbers	in	Limit of Mach Error
Initial_Mach_Feedback	Mach_Numbers	in	Mach Feedback from missile at startup
Initial_Mach_Error_Gain	Mach_Number_Gain	in	Gain to convert from mach error to raw throttle command
Initial_Mach_Error_Integral_Limit	Throttle_Commands	in	Limit for Mach Error Integral at startup
Initial_Throttle_Command	Throttle_Commands	in	Throttle Command at startup
Initial_Throttle_Command_Rate_Limit	Throttle_Commands	in	Limit on Throttle Command Rate at startup
Initial_Lower_Throttle_Command_Limit	Throttle_Commands	in	Lower Limit of Throttle Command
Initial_Upper_Throttle_Command_Limit	Throttle_Commands	in	Upper Limit of Throttle Command
Initial_Throttle_Bandwidth	Throttle_Commands	in	3 db bandwidth of the throttle command

Subprograms: The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies a Mach_Number by a Mach_Number_Gain yielding a Throttle_Command

3.6.7.2.9.1.3 LOCAL ENTITIES

Packages:

The packages Integral_Plus_Proportional_Gain, Tustin_Integrator_With_Limit, Tustin_Integrator_With_Asymmetric_Limit, and Absolute_Limiter are instantiated inside the package body.

3.6.7.2.9.1.4 INTERRUPTS

None.

3.6.7.2.9.1.5 TIMING AND SEQUENCING

None.

3.6.7.2.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.7.2.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Compute_Throttle_Command	function	Computes a new Throttle Command
Update_Mach_Error_Limit	procedure	Updates the Mach Error Limit
Update_Mach_Error_Integral_Limit	procedure	Updates the Mach Error Integral Limit
Update_Throttle_Rate_Limit	procedure	Updates the Throttle Rate Limit
Update_Throttle_Command_Limits	procedure	Updates the Upper and Lower Throttle Command limits
Update_Mach_Error_Gain	procedure	Updates the Mach Error Gain
Update_Throttle_Bandwidth	procedure	Updates the 3 db bandwidth which the throttle rate can lie inside

3.6.7.2.9.1.8 PART DESIGN

None.

```
package Fuel_Control_Parts is
                                                              pragma PAGE;
   generic
      type Mach_Numbers          is digits <>;
      type Mach_Number_Gains     is digits <>;
      type Throttle_Commands     is digits <>;

      Initial_Mach_Command                    : Mach_Numbers;
      Initial_Mach_Error_Limit                : Mach_Numbers;
      Initial_Mach_Feedback                   : Mach_Numbers;
      Initial_Mach_Error_Gain                 : Mach_Number_Gains;
      Initial_Mach_Error_Integral_Limit       : Throttle_Commands;
      Initial_Throttle_Command                : Throttle_Commands;
      Initial_Throttle_Command_Rate_Limit     : Throttle_Commands;
      Initial_Lower_Throttle_Command_Limit    : Throttle_Commands;
      Initial_Upper_Throttle_Command_Limit    : Throttle_Commands;
      Initial_Throttle_Bandwidth              : Throttle_Commands;

      with function "*" (Left  : Mach_Numbers;
                         Right : Mach_Number_Gains)
                        return Throttle_Commands is <>;

   package Throttle_Command_Manager is

      function Compute_Throttle_Command
                              (Mach_Command  : in Mach_Numbers;
                               Mach_Feedback : in Mach_Numbers)
                           return Throttle_Commands;

      procedure Update_Mach_Error_Limit
                              (New_Limit : in Mach_Numbers);

      procedure Update_Mach_Error_Integral_Limit
                              (New_Limit : in Throttle_Commands);

      procedure Update_Throttle_Rate_Limit
                              (New_Limit : in Throttle_Commands);

      procedure Update_Throttle_Command_Limits
                              (New_Lower_Limit : in Throttle_Commands;
                               New_Upper_Limit : in Throttle_Commands);

      procedure Update_Mach_Error_Gain
                              (New_Gain : in Mach_Number_Gains);

      procedure Update_Throttle_Bandwidth
                              (New_Bandwidth : in Throttle_Commands);

   end Throttle_Command_Manager;

end Fuel_Control_Parts;
```

(This page left intentionally blank.)

3.6.8 MATHEMATICAL

(This page left intentionally blank.)

3.6.8.1 COORDINATE_VECTOR_MATRIX_ALGEBRA TLCSC (CATALOG #P45-0)

This part consists of generic packages and functions which define and/or operate on coordinate vectors and matrices. A coordinate vector is a three-element array. A coordinate matrix is a 3 x 3 array. These arrays are dimensioned with scalar types defined by the user.

WARNING: The units in this part ASSUME the axes types used to dimension the arrays have a length of 3. If they do not, the units will not function properly. No length checks are performed by the units.

3.6.8.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Type	Requirements Allocation
Vector_Operations	generic package	R024, R050, R051, R052
Matrix_Operations	generic package	R070, R071, R060, R067, R072 R078
Vector_Scalar_Operations	generic package	R054, R055
Matrix_Scalar_Operations	generic package	R056, R057
Cross_Product	generic function	R053
Matrix_Vector_Multiply	generic function	R049
Matrix_Matrix_Multiply	generic function	R068

3.6.8.1.2 INPUT/OUTPUT

None.

3.6.8.1.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.8.1.4 LOCAL ENTITIES

None.

3.6.8.1.5 INTERRUPTS

None.

3.6.8.1.6 TIMING AND SEQUENCING

None.

3.6.8.1.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.1.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Vector_Operations	generic package	Defines a vector type and provides general operations on that type
Matrix_Operations	generic package	Defines a matrix type and provides general operations on that type
Vector_Scalar_Operations	generic package	Provides operations to multiply and divide each element of a vector by a scalar
Matrix_Scalar_Operations	generic package	Provides operations to multiply and divide each element of a matrix by a scalar
Cross_Product	generic function	Performs cross-product operations on two 3-dimensional vectors
Matrix_Vector_Multiply	generic function	Multiplies a 3 x 3 matrix by a 3 x 1 vector, returning a 3 x 1 vector c(i) := a(i,j) * b(j)
Matrix_Matrix_Multiply	generic function	Multiples two 3 x 3 matrices, returning the resultant matrix c(i,j) := a(i,k) * b(k,j)

3.6.8.1.9 PART DESIGN

3.6.8.1.9.1 VECTOR_OPERATIONS (CATALOG #P46-0)

Taking the generic formal parameters Elements and Axes, this package defines a vector as a one-dimensional array of these elements. It then defines operations on the vector. The operations provided are described in the decomposition section.

3.6.8.1.9.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
"+"	R050
"-"	R051
Vector_Length	R052
Dot_Product	R208
Sparse_Right_Z_Add	R205
Sparse_Right_X_Add	R206
Sparse_Right_XY_Subtract	R207
Set_to_Zero_Vector	

3.6.8.1.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Axes	scalar type	Used to dimension the exported vector type
Elements	floating point type	Data type of elements in exported vector type
Elements_Squared	floating point type	Data type resulting from multiplying two objects of type Elements

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Elements * Elements := Elements_Squared
Sqrt	function	Square root operator

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Vectors	N/A	See decomposition section	One-dimensional array of Elements

3.6.8.1.9.1.3 LOCAL ENTITIES

None.

3.6.8.1.9.1.4 INTERRUPTS

None.

3.6.8.1.9.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Coordinate_Vector_Matrix_Algebra;
...
   package CVMA renames Coordinate_Vector_Matrix_Algebra;

   type My_Axes              is (x, y, z);
   type My_Elements          is new FLOAT;
   type My_Elements_Squared is new My_Elements;
   ...
   function "*" (Left  : My_Elements;
                 Right : My_Elements) return My_Elements_Squared;
   ...
   function SqRt (Left : My_Elements_Squared) return My_Elements;
   ...
   package VOpns is new
          CVMA.Vector_Operations
             (Axes             => My_Axes,
              Elements         => My_Elements,
              Elements_Squared => My_Elements_Squared);
   use VOpns;
   ...
   Vector1 : VOpns.Vectors;
   Vector2 : VOpns.Vectors;
   ...
   begin
      ...
      Vector1 := Vector1 + Vector2;
```

3.6.8.1.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.1.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"+"	function	Adds two vectors of the same type returning a resultant vector c(i) := a(i) + b (i)
"-"	function	Subtracts two vectors of the same type returning a resultant vector c(i) := a(i) - b (i)
Dot_Product	function	Calculates the dot product of two vectors returning the result c := a(i) * b(i)
Vector_Length	function	Computes the length of a vector returning the result c := Sqrt(sum of a(i)**2)
Sparse_Right_Z_Add	function	Adds two vectors, assuming the third element of the second vector equals 0
Sparse_Right_X_Add		Adds two vectors, assuming the first element of the second vector equals 0
Sparse_Right_XY_Subtract	function	Subtracts two vectors, assuming the first and second elements of second vector equals 0
Set_to_Zero_Vector	function	Sets all elements of a vector equal to 0

The following table summarizes the allocation of catalog numbers to this part:

Name	Catalog #
"+"	P703-0
"-"	P704-0
Vector_Length	P705-0
Dot_Product	P706-0
Sparse_Right_Z_Add	P707-0
Sparse_Right_X_Add	P708-0
Sparse_Right_XY_Subtract	P709-0
Set_to_Zero_Vector	P710-0

3.6.8.1.9.1.8 PART DESIGN

None.

3.6.8.1.9.2 MATRIX_OPERATIONS (CATALOG #P47-0)

Taking the generic formal parameters Elements and Axes, this package defines a matrix as a two-dimensional array of these elements. It then defines operations on the matrix. The operations provided are shown in the decomposition section.

3.6.8.1.9.2.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirement Allocation
"+" (matrices + matrices)	R070
"-" (matrices - matrices)	R071
"+" (matrices + elements)	R060
"-" (matrices - elements)	R067
Set_To_Identity_Matrix	R072
Set_To_Zero_Matrix	R078

3.6.8.1.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Axes	scalar type	Used to dimension exported matrix type
Elements	floating point type	Data type of elements in exported matrix type

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Matrices	N/A	See decomposition section	Two-dimensional array of Elements

3.6.8.1.9.2.3 LOCAL ENTITIES

None.

3.6.8.1.9.2.4 INTERRUPTS

None.

3.6.8.1.9.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Coordinate_Vector_Matrix_Algebra;
...
   package CVMA renames Coordinate_Vector_Matrix_Algebra;

   type My_Axes            is (x, y, z);
   type My_Elements        is new FLOAT;
   ...
   package MOpns is new
           CVMA.Matrix_Operations
              (Axes => My_Axes,
               Elements => My_Elements);
   use MOpns;
   ...
   Matrix1 : MOpns.Matrixs;
   Matrix2 : MOpns.Matrixs;
   ...
   begin
      ...
      Matrix1 := Matrix1 + Matrix2;
```

3.6.8.1.9.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.1.9.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"+" (matrices + matrices)	function	Adds two 3x3 matrices c(i,j) := a(i,j) + b(i,j)
"-" (matrices - matrices)	function	Subtracts two 3x3 matrices c(i,j) := a(i,j) - b(i,j)
"+" (matrices + elements)	function	Adds a scalar value to each element of a 3x3 matrix c(i,j) := a(i,j) + b
"-" (matrices - elements)	function	Subtracts a scalar value from each element of a 3x3 matrix c(i,j) := a(i,j) - b
Set_To_Identity_Matrix	function	Initializes a 3x3 matrix to an identity matrix
Set_To_Zero_Matrix	function	Sets each element of a 3x3 matrix to zero

The following table summarizes the allocation of catalog numbers to this part:

Name	Catalog #
"+" (matrices + matrices)	P711-0
"-" (matrices - matrices)	P712-0
"+" (matrices + elements)	P713-0
"-" (matrices - elements)	P714-0
Set_to_Identity_Matrix	P715-0
Set_to_Zero_Matrix	P716-0

3.6.8.1.9.2.8 PART DESIGN

None.

3.6.8.1.9.3 VECTOR_SCALAR_OPERATIONS (CATALOG #P49-0)

This package provides the functions to allow the user to multiply or divide each element of a vector by a scalar. In addition, a multiplication function is provided for a sparse vector.

3.6.8.1.9.3.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirement Allocation
"*"	R054
Sparse_X_Vector_Scalar_Multiply	R209
"/"	R055

3.6.8.1.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Axes	scalar type	Used to dimension imported vector types
Elements1	floating point type	Type of elements on Vectors1
Elements2	floating point type	Type of elements on Vectors2
Scalars	floating point type	Data type of scale factors

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator used to define the operation: Elements1 * Scalars := Elements2
"/"	function	Division operator used to define the operation: Elements2 / Scalars := Elements1

3.6.8.1.9.3.3 LOCAL ENTITIES

None.

3.6.8.1.9.3.4 INTERRUPTS

None.

3.6.8.1.9.3.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Coordinate_Vector_Matrix_Algebra;
...
   package CVMA renames Coordinate_Vector_Matrix_Algebra;

   type My_Axes                is (x, y, z);
   type My_Elements1           is new FLOAT;
   type My_Elements1_Squared is new My_Elements1;
   type My_Elements2           is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Scalars             is new FLOAT;
   ...
   function "*" (Left  : My_Elements1;
                 Right : My_Elements1) return My_Elements1_Squared;
   function "*" (Left  : My_Elements2;
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function "*" (Left  : My_Elements1;
                 Right : Scalars) return My_Elements2;
   function "/" (Left  : My_Elements2;
                 Right : Scalars) return My_Elements1;
   ...
   function SqRt (Left : My_Elements1_Squared) return My_Elements1;
   function SqRt (Left : My_Elements2_Squared) return My_Elements2;
   ...
   package VOpns1 is new
          CVMA.Vector_Operations
             (Axes              => My_Axes,
              Elements          => My_Elements1,
```

```
              Elements_Squared => My_Elements1_Squared);
   use VOpns1;
   ...
   package VOpns2 is new
           CVMA.Vector_Operations
              (Axes             => My_Axes,
               Elements         => My_Elements2,
               Elements_Squared => My_Elements2_Squared);
   use VOpns2;
   ...
   package VSOpns is new
           CVMA.Vector_Scalar_Operations
              (Axes      => My_Axes,
               Elements1 => My_Elements1,
               Elements2 => My_Elements2,
               Scalars   => My_Scalars,
               Vectors1  => VOpns1.Vectors,
               Vectors2  => VOpns2.Vectors);
   use VSOpns;
   ...
   Temp    : My_Scalars;
   Vector1 : VOpns1.Vectors;
   Vector2 : VOpns2.Vectors;
   ...
   begin
      ...
      Vector2 := Vector1 * Temp;
```

3.6.8.1.9.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.1.9.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies each element of a 3x1 vector by a scalar value c(i) := a(i) * b
Sparse_X_Vector_Scalar_Multiply	function	Multiplies each element of a 3x1 vector by a scalar value assuming the first element of the vector equals 0
"/"	function	Divides each element of a 3x1 vector by a scalar value c(i) := a(i) / b

The following table summarizes the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P717-0
Sparse_X_Vector_Scalar_Multiply	P718-0
"/"	P719-0

3.6.8.1.9.3.8 PART DESIGN

None.

3.6.8.1.9.4 MATRIX_SCALAR_OPERATIONS (CATALOG #P49-0)

This package provides the functions to allow the user to multiply or divide each element of a matrix by a scalar.

3.6.8.1.9.4.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirement Allocation
"*"	R056
"/"	R057

3.6.8.1.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Axes	scalar type	Used to dimension incoming matrix types
Elements1	floating point type	Type of elements in Matrices1
Elements2	floating point type	Type of elements in Matrices2
Matrices1	array	Two-dimensional array of Elements1
Matrices2	array	Two-dimensional array of Elements2

Subprograms:

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator used to define the operation: Elements1 * Scalars := Elements2
"/"	function	Division operator used to define the operation: Elements2 / Scalars := Elements1

3.6.8.1.9.4.3 LOCAL ENTITIES

None.

3.6.8.1.9.4.4 INTERRUPTS

None.

3.6.8.1.9.4.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Coordinate_Vector_Matrix_Algebra;
...
   package CVMA renames Coordinate_Vector_Matrix_Algebra;

   type My_Axes                is (x, y, z);
   type My_Elements1           is new FLOAT;
   type My_Elements2           is new FLOAT;
   type My_Scalars             is new FLOAT;
   ...
   function "*" (Left  : My_Elements1;
                 Right : Scalars) return My_Elements2;
   function "/" (Left  : My_Elements2;
                 Right : Scalars) return My_Elements1;
   ...
   package MOpns1 is new
          CVMA.Matrix_Operations
             (Axes             => My_Axes,
              Elements         => My_Elements1);
   use MOpns1;
   ...
   package MOpns2 is new
          CVMA.Matrix_Operations
             (Axes             => My_Axes,
              Elements         => My_Elements2);
   use MOpns2;
   ...
   package MSOpns is new
          CVMA.Matrix_Scalar_Operations
```

```
               (Axes      => My_Axes,
                Elements1 => My_Elements1,
                Elements2 => My_Elements2,
                Matrices1 => MOpns1.Matrices,
                Matrices2 => MOpns2.Matrices);
   ...
   Temp    : My_Scalars;
   Matrix1 : MOpns1.Matrices;
   Matrix2 : MOpns2.Matrices;
   ...
   begin
      ...
      Matrix2 := Matrix1 * Temp;
```

3.6.8.1.9.4.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.1.9.4.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies each element of a 3x3 matrix by a scalar value c(i,j) := a(i,j) * b
"/"	function	Divides each element of a 3x3 matrix by a scalar value c(i,j) := a(i,j) / b

The following table summarizes the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P720-0
"/"	P721-0

3.6.8.1.9.4.8 PART DESIGN

None.

3.6.8.1.9.5 CROSS_PRODUCT (CATALOG #P50-0)

This generic function performs a cross product operation on two vectors, returning the resultant vector. The three vectors are each three-dimensional, coordinate vectors. None of the three need to contain the same type of elements.

3.6.8.1.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R053.

3.6.8.1.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Axes	scalar type	Used to dimension imported vector types
Left_Elements	floating point type	Data type of elements in left input vector
Right_Elements	floating point type	Data type of elements in right input vector
Result_Elements	floating point type	Data type of elements in result vector
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector
Result_Vectors	array	Data type of result vector

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator used to define the operation: Left_Elements * Right_Elements := Result_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Vectors	N/A	Matrix to be used on the left side of the cross-product operation
Right	Right_Vectors	N/A	Matrix to be used on the right side of the cross-product operation

3.6.8.1.9.5.3 INTERRUPTS

None.

3.6.8.1.9.5.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Coordinate_Vector_Matrix_Algebra;
...
   package CVMA renames Coordinate_Vector_Matrix_Algebra;

   type My_Axes              is (x, y, z);
   type My_Elements1         is new FLOAT;
   type My_Elements1_Squared is new My_Elements1;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   type My_Elements3_Squared is new My_Elements3;
   ...
   function "*" (Left  : My_Elements1;
                 Right : My_Elements1) return My_Elements1_Squared;
   function "*" (Left  : My_Elements2;
                 Right : My_Elements2) return My_Elements2_Squared;
   function "*" (Left  : My_Elements3;
                 Right : My_Elements3) return My_Elements3_Squared;
   ...
   function "*" (Left  : My_Elements1;
                 Right : My_Elements2) return My_Elements3;
   ...
   function SqRt (Left : My_Elements1_Squared) return My_Elements1;
   function SqRt (Left : My_Elements2_Squared) return My_Elements2;
   function SqRt (Left : My_Elements3_Squared) return My_Elements3;
   ...
   package VOpns1 is new
           CVMA.Vector_Operations
              (Axes             => My_Axes,
               Elements         => My_Elements1,
               Elements_Squared => My_Elements1_Squared);
   use VOpns1;
   ...
   package VOpns2 is new
           CVMA.Vector_Operations
              (Axes             => My_Axes,
               Elements         => My_Elements2,
               Elements_Squared => My_Elements2_Squared);
   use VOpns2;
   ...
   package VOpns3 is new
           CVMA.Vector_Operations
              (Axes             => My_Axes,
               Elements         => My_Elements3,
               Elements_Squared => My_Elements3_Squared);
   use VOpns3;
   ...
   function Cross_Product is new
```

```
         CVMA.Cross_Product
            (Axes             => My_Axes,
             Left_Elements    => My_Elements1,
             Right_Elements   => My_Elements2,
             Result_Elements  => My_Elements3,
             Left_Vectors     => VOpns1.Vectors,
             Right_Vectors    => VOpns2.Vectors,
             Result_Vectors   => VOpns3.Vectors);
   ...
   Vector1 : VOpns1.Vectors;
   Vector2 : VOpns2.Vectors;
   Vector3 : VOpns3.Vectors;
   ...
   begin
      ...
      Vector3 := Cross_Product(Vector1, Vector2);
```

3.6.8.1.9.5.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.1.9.5.6 DECOMPOSITION

None.

3.6.8.1.9.6 MATRIX_VECTOR_MULTIPLY (CATALOG #P51-0)

This generic function allows the user to multiply a 3 x 3 matrix by a 3 x 1 vector, returning the resultant 3 x 1 vector. Both the matrix and the vector contain elements in the x, y, and z axes of the Cartesian coordinate system.

3.6.8.1.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R049.

3.6.8.1.9.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Axes	scalar type	Used to dimension incoming array types
Input_Vector_Elements	floating point type	Data type of elements in input vector
Output_Vector_Elements	floating point type	Data type of elements in output vector
Matrix_Elements	floating point type	Data type of elements in input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector
Matrices	array	Data type of input matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Matrix_Elements * Input_Vector_Elements := Output_Vector_Elements

FORMAL PARAMETERS:

The following chart describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be used in calculations
Vector	Input_Vectors	In	Vector to be used in calculations

3.6.8.1.9.6.3 INTERRUPTS

None.

3.6.8.1.9.6.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Coordinate_Vector_Matrix_Algebra;
...
   package CVMA renames Coordinate_Vector_Matrix_Algebra;

   type My_Axes                is (x, y, z);
   type My_Elements1           is new FLOAT;
   type My_Elements1_Squared is new My_Elements1;
```

```
type My_Elements2         is new FLOAT;
type My_Elements2_Squared is new My_Elements2;
type My_Elements3         is new FLOAT;
type My_Elements3_Squared is new My_Elements3;
...
function "*" (Left  : My_Elements2;
              Right : My_Elements2) return My_Elements2_Squared;
function "*" (Left  : My_Elements3;
              Right : My_Elements3) return My_Elements3_Squared;
...
function "*" (Left  : My_Elements1;
              Right : My_Elements2) return My_Elements3;
...
function SqRt (Left : My_Elements2_Squared) return My_Elements2;
function SqRt (Left : My_Elements3_Squared) return My_Elements3;
...
package MOpns1 is new
        CVMA.Matrix_Operations
           (Axes              => My_Axes,
            Elements          => My_Elements1);
use MOpns1;
...
package VOpns2 is new
        CVMA.Vector_Operations
           (Axes              => My_Axes,
            Elements          => My_Elements2,
            Elements_Squared  => My_Elements2_Squared);
use VOpns2;
...
package VOpns3 is new
        CVMA.Vector_Operations
           (Axes              => My_Axes,
            Elements          => My_Elements3,
            Elements_Squared  => My_Elements3_Squared);
use VOpns3;
...
function "*" is new
        CVMA.Matrix_Vector_Multiply
           (Axes                   => My_Axes,
            Input_Vector_Elements  => My_Elements2,
            Output_Vector_Elements => My_Elements3,
            Matrix_Elements        => My_Elements1,
            Matrices               => MOpns1.Matrices,
            Input_Vectors          => VOpns2.Vectors,
            Output_Vectors         => VOpns3.Vectors);
...
Matrix1 : MOpns1.Matrices;
Vector2 : VOpns2.Vectors;
Vector3 : VOpns3.Vectors;
...
begin
   ...
   Vector3 := Matrix1 * Vector2;
```

3.6.8.1.9.6.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.1.9.6.6 DECOMPOSITION

None.

3.6.8.1.9.7 MATRIX_MATRIX_MULTIPLY (CATALOG #P52-0)

This generic function allows the user to multiply two 3 x 3 matrix, returning the resultant 3 x 3 matrix. Both matrices contain elements in the x, y, and z axes of the Cartesian coordinate system.

3.6.8.1.9.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R068.

3.6.8.1.9.7.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Axes	scalar type	Used to dimension imported array types
Left_Elements	floating point type	Data type of elements in left input matrix
Right_Elements	floating point type	Data type of elements in right input matrix
Result_Elements	floating point type	Data type of elements in output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Result_Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Left_Elements * Right_Elements := Result_Elements

FORMAL PARAMETERS:

The following chart describes this part's formal parameters:

Name	Type	Mode	Description
Matrix1	Left_Matrices	In	First matrix used for multiplication operation
Matrix2	Right_Matrices	In	Second matrix used for multiplication operation

3.6.8.1.9.7.3 INTERRUPTS

None.

3.6.8.1.9.7.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Coordinate_Vector_Matrix_Algebra;
...
   package CVMA renames Coordinate_Vector_Matrix_Algebra;

   type My_Axes               is (x, y, z);
   type My_Elements1          is new FLOAT;
   type My_Elements2          is new FLOAT;
   type My_Elements3          is new FLOAT;
   ...
   function "*" (Left  : My_Elements1;
                 Right : My_Elements2) return My_Elements3;
   ...
   package MOpns1 is new
           CVMA.Matrix_Operations
              (Axes            => My_Axes,
               Elements        => My_Elements1);
   use MOpns1;
   ...
   package MOpns2 is new
           CVMA.Matrix_Operations
              (Axes            => My_Axes,
               Elements        => My_Elements2);
   use MOpns2;
   ...
   package MOpns3 is new
           CVMA.Matrix_Operations
              (Axes            => My_Axes,
               Elements        => My_Elements3);
   use MOpns31;
   ...
   function "*" is new
           CVMA.Matrix_Matrix_Multiply
              (Axes          => My_Axes,
               Left_Elements  => My_Elements1,
```

```
                Right_Elements  => My_Elements1,
                Result_Elements => My_Elements3,
                Left_Matrices   => MOpns1.Matrices,
                Right_Matrices  => MOpns2.Matrices,
                Output_Matrices => MOpns3.Matrices);
   ...
   Matrix1 : MOpns1.Matrices;
   Matrix2 : MOpns2.Matrices;
   Matrix3 : MOpns3.Matrices;
   ...
   begin
      ...
      Matrix3 := Matrix1 * Matrix2;
```

3.6.8.1.9.7.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.1.9.7.6 DECOMPOSITION

None.

(This page left intentionally blank.)

```
package Coordinate_Vector_Matrix_Algebra is

pragma PAGE;
   generic
     type Axes              is (<>);
     type Elements          is digits <>;
     type Elements_Squared is digits <>;
     with function "*" (Left  : Elements;
                         Right : Elements) return Elements_Squared is <>;
     with function Sqrt (Input : Elements_Squared) return Elements is <>;
   package Vector_Operations is

     type Vectors is array(Axes) of Elements;

     function "+" (Left  : Vectors;
                   Right : Vectors) return Vectors;

     function "-" (Left  : Vectors;
                   Right : Vectors) return Vectors;

     function Vector_Length (Vector : Vectors) return Elements;

     function Dot_Product (Vector1 : Vectors;
                           Vector2 : Vectors) return Elements_Squared;

     function Sparse_Right_Z_Add (Left  : Vectors;
                                  Right : Vectors) return Vectors;

     function Sparse_Right_X_Add (Left  : Vectors;
                                  Right : Vectors) return Vectors;

     function Sparse_Right_Xy_Subtract (Left  : Vectors;
                                        Right : Vectors) return Vectors;

     function Set_To_Zero_Vector return Vectors;

   end Vector_Operations;

pragma PAGE;
   generic
     type Axes     is (<>);
     type Elements is digits <>;
   package Matrix_Operations is

     type Matrices is array (Axes, Axes) of Elements;

     function "+" (Left : Matrices;
                  Right : Matrices) return Matrices;

     function "-" (Left  : Matrices;
                   Right : Matrices) return Matrices;

     function "+" (Matrix : Matrices;
                   Addend : Elements) return Matrices;

     function "-" (Matrix     : Matrices;
                   Subtrahend : Elements) return Matrices;
```

```
      function Set_To_Identity_Matrix return Matrices;

      function Set_To_Zero_Matrix return Matrices;

   end Matrix_Operations;

pragma PAGE;
   generic
      type Axes      is (<>);
      type Elements1 is digits <>;
      type Elements2 is digits <>;
      type Scalars   is digits <>;
      type Vectors1  is array(Axes) of Elements1;
      type Vectors2  is array(Axes) of Elements2;
      with function "*" (Left  : Elements1;
                         Right : Scalars) return Elements2 is <>;
      with function "/" (Left  : Elements2;
                         Right : Scalars) return Elements1 is <>;
   package Vector_Scalar_Operations is

      function "*" (Vector     : Vectors1;
                    Multiplier : Scalars) return Vectors2;

      function Sparse_X_Vector_Scalar_Multiply
                   (Vector     : Vectors1;
                    Multiplier : Scalars) return Vectors2;

      function "/" (Vector  : Vectors2;
                    Divisor : Scalars) return Vectors1;

   end Vector_Scalar_Operations;

pragma PAGE;
   generic
      type Axes      is (<>);
      type Elements1 is digits <>;
      type Elements2 is digits <>;
      type Scalars   is digits <>;
      type Matrices1 is array (Axes, Axes) of Elements1;
      type Matrices2 is array (Axes, Axes) of Elements2;
      with function "*" (Left  : Elements1;
                         Right : Scalars) return Elements2 is <>;
      with function "/" (Left  : Elements2;
                         Right : Scalars) return Elements1 is <>;
   package Matrix_Scalar_Operations is

      function "*" (Matrix     : Matrices1;
                    Multiplier : Scalars) return Matrices2;

      function "/" (Matrix  : Matrices2;
                    Divisor : Scalars) return Matrices1;

   end Matrix_Scalar_Operations;

pragma PAGE;
   generic
```

```
      type Axes            is (<>);
      type Left_Elements   is digits <>;
      type Right_Elements  is digits <>;
      type Result_Elements is digits <>;
      type Left_Vectors    is array(Axes) of Left_Elements;
      type Right_Vectors   is array(Axes) of Right_Elements;
      type Result_Vectors  is array(Axes) of Result_Elements;
      with function "*" (Left  : Left_Elements;
                         Right : Right_Elements) return Result_Elements is <>;
   function Cross_Product (Left  : Left_Vectors;
                           Right : Right_Vectors) return Result_Vectors;

pragma PAGE;
   generic
      type Axes                   is (<>);
      type Input_Vector_Elements  is digits <>;
      type Output_Vector_Elements is digits <>;
      type Matrix_Elements        is digits <>;
      type Input_Vectors          is array (Axes) of Input_Vector_Elements;
      type Output_Vectors         is array (Axes) of Output_Vector_Elements;
      type Matrices               is array (Axes, Axes) of Matrix_Elements;
      with function "*" (Left  : Matrix_Elements;
                         Right : Input_Vector_Elements)
                        return Output_Vector_Elements is <>;
   function Matrix_Vector_Multiply
              (Matrix : Matrices;
               Vector : Input_Vectors) return Output_Vectors;

pragma PAGE;
   generic
      type Axes            is (<>);
      type Left_Elements   is digits <>;
      type Right_Elements  is digits <>;
      type Result_Elements is digits <>;
      type Left_Matrices   is array (Axes, Axes) of Left_Elements;
      type Right_Matrices  is array (Axes, Axes) of Right_Elements;
      type Result_Matrices is array (Axes, Axes) of Result_Elements;
      with function "*" (Left  : Left_Elements;
                         Right : Right_Elements) return Result_Elements is <>;
   function Matrix_Matrix_Multiply
              (Matrix1 : Left_Matrices;
               Matrix2 : Right_Matrices) return Result_Matrices;

end Coordinate_Vector_Matrix_Algebra;
```

(This page left intentionally blank.)

3.6.8.2 GENERAL_VECTOR_MATRIX_ALGEBRA (SPEC) TLCSC (CATALOG #P167-0)

This part is a package of generic packages and generic functions. The LLCSC's take two different forms. One form defines vector and matrix types, along with general operations on these types. The other form requires that vector and matrix types be provided as generic parameters and performs operations on data objects of different types.

Many of the parts have both an unconstrained and constrained or restricted and unrestricted versions. The constrained/restricted versions of these parts are less flexible in the dimensioning of the input arrays, but require fewer internal calculations.

The generic functions/package which import generic formal array types have been designed to work in conjunction with the data types exported by the generic packages.

3.6.8.2.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
General_Vector_Matrix_Algebra	R058
Vector_Operations_Unconstrained	R061, R062, R063, R104
Vector_Operations_Constrained	R061, R062, R063, R104
Matrix_Operations_Unconstrained	R075, R076, R079, R080, R155, R156
Matrix_Operations_Constrained	R075, R076, R079, R080, R155, R156
Dynamically_Sparse_Matrix_Operations_Unconstrained	R226
Dynamically_Sparse_Matrix_Operations_Constrained	R226
Symmetric_Half_Storage_Matrix_Operations	R211
Symmetric_Full_Storage_Matrix_Operations_Unconstrained	R227
Symmetric_Full_Storage_Matrix_Operations_Constrained	R227
Diagonal_Matrix_Operations	R212
Vector_Scalar_Operations_Unconstrained	R065, R066
Vector_Scalar_Operations_Constrained	R065, R066
Matrix_Scalar_Operations_Unconstrained	R073, R074
Matrix_Scalar_Operations_Constrained	R073, R074
Diagonal_Matrix_Scalar_Operations	R212
Matrix_Vector_Multiply_Unrestricted	R069
Matrix_Vector_Multiply_Restricted	R069
Vector_Matrix_Multiply_Unrestricted	
Vector_Matrix_Multiply_Restricted	
Vector_Vector_Transpose_Multiply_Unrestricted	
Vector_Vector_Transpose_Multiply_Restricted	
Matrix_Matrix_Multiply_Unrestricted	R077
Matrix_Matrix_Multiply_Restricted	R077
Matrix_Matrix_Transpose_Multiply_Unrestricted	
Matrix_Matrix_Transpose_Multiply_Restricted	
Dot_Product_Operation_Unrestricted	R063
Dot_Product_Operation_Restricted	R063
Diagonal_Full_Matrix_Add_Unrestricted	R212
Diagonal_Full_Matrix_Add_Restricted	R212
ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix	
ABA_Trans_Vector_Sq_Matrix	
ABA_Trans_Vector_Scalar	
Column_Matrix_Operations	

Allocation of Parts to General Vector/Matrix Algebra TLCSC

3.6.8.2.2 INPUT/OUTPUT

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following table describes the exceptions exported by this part:

Name	Description
Dimension_Error	Raised by a routine or package when input received has dimensions incompatible with the type of operation to be performed

3.6.8.2.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.8.2.4 LOCAL ENTITIES

None.

3.6.8.2.5 INTERRUPTS

None.

3.6.8.2.6 TIMING AND SEQUENCING

None.

3.6.8.2.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.2.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Vector_Operations_ Unconstrained	generic package	Defines an unconstrained vector type and provides general operations on that type
Vector_Operations_ Constrained	generic package	Defines a constrained vector type and provides general operations on that type
Matrix_Operations_ Unconstrained	generic package	Defines an unconstrained matrix type and provides general operations on that type
Matrix_Operations_ Constrained	generic package	Defines a constrained matrix type and provides general operations on that type
Dynamically_Sparse_Matrix_ Operations_Unconstrained	generic package	Defines an unconstrained matrix type which is dynamically sparse and provides general operations on that type
Dynamically_Sparse_Matrix_ Operations_Constrained	generic package	Defines a constrained matrix type which is dynamically sparse and provides general operations on that type
Symmetric_Half_Storage_ Matrix_Operations	generic package	Defines a constrained symmetric, half storage matrix in which only the bottom half of the matrix is stored and provides general operations on that type
Symmetric_Full_Storage_ Matrix_Operations_ Unconstrained	generic package	Defines an unconstrained symmetric full storage matrix type and provides general operations on that type
Symmetric_Full_Storage_ Matrix_Operations_ Constrained	generic package	Defines a constrained symmetric full storage matrix type and provides general operations on that type
Diagonal_Matrix_Operations	generic package	Defines a constrained diagonal matrix type where only the diagonal elements are stored and provides general operations on that type
Vector_Scalar_Operations_ Unconstrained	generic package	Provides operations to multiply and divide an unconstrained vector by a scalar
Vector_Scalar_Operations_ Constrained	generic package	Provides operations to multiply and divide a constrained vector by a scalar
Matrix_Scalar_Operations_ Unconstrained	generic package	Provides operations to multiply and divide an unconstrained matrix by a scalar
Matrix_Scalar_Operations_ Constrained	generic package	Provides operations to multiply and divide a constrained matrix by a scalar
Diagonal_Matrix_Scalar_ Operations	generic package	Provides operations to multiply and divide a diagonal matrix by a scalar
Matrix_Vector_Multiply_	generic	Multiplies an m x n matrix by an

Unrestricted	package	n x 1 vector, returning an m x 1 vector c(i,j) = a(i) * b(j)
Matrix_Vector_Multiply_Restricted	generic function	Multiplies an m x n matrix by an n x 1 vector, returning an m x 1 vector c(i,j) = a(i) * b(j)
Vector_Matrix_Multiply_Unrestricted	generic package	Multiplies a 1 x m vector by an m x n matrix, returning a 1 x n vector c(j) = a(i) * b(i,j)
Vector_Matrix_Multiply_Restricted	generic function	Multiplies a 1 x m vector by an m x n matrix, returning a 1 x n vector c(j) = a(i) * b(i,j)
Vector_Vector_Transpose_Multiply_Unrestricted	generic package	Multiplies an m x 1 vector by the transpose of an n x 1 vector, returning an m x n matrix
Vector_Vector_Transpose_Multiply_Restricted	generic function	Multiplies an m x 1 vector by the transpose of an n x 1 vector, returning an m x n matrix
Unrestricted	package	n x p matrix, returning an m x p matrix
Matrix_Matrix_Multiply_Restricted	generic function	Multiplies an m x n matrix by an n x p amtrix, returning an m x p matrix c(i,j) = a(i,k) * b(k,j)
Matrix_Matrix_Transpose_Multiply_Unrestricted	generic package	Multiplies an m x n matrix by the transpose of a p x n matrix, returning an m x p matrix
Matrix_Matrix_Transpose_Multiply_Restricted	generic function	Multiplies an m x n matrix by the transpose of a p x n matrix, returning an m x p matrix c(i,j) = a(i,k) * b(j,k)
Dot_Product_Operation_Unrestricted	generic package	Performs a dot product operation on two unconstrained vectors c = a(i) * b(i)
Dot_Product_Operation_Restricted	generic function	Performs a dot product operation on two constrained vectors c = a(i) * b(i)
Diagonal_Full_Matrix_Add_Unrestricted	generic package	Adds an m x m diagonal matrix to an unconstrained, m x m full storage matrix c(i) = a(i) + b(i)
Diagonal_Full_Matrix_Add_Restricted	generic function	Adds an m x m diagonal matrix to a constrained, m x m full storage matrix c(i) = a(i) + b(i)
ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix	generic package	Does an ABA transpose to dynamically sparse matrix (m x n) and a square matrix (n x n)
ABA_Trans_Vector_Sq_Matrix	generic package	Does an ABA transpose to a vector (1 x m) and a square matrix (m x m)
ABA_Trans_Vector_Scalar	generic	Does an ABA transpose to a vector

	package	(1 x m) and a scalar
ABA_Trans_Col_Matrix_Sq_Matrix	generic package	Does an ABA transpose to a column matrix (m x n) and a square matrix (n x n)
Column_Matrix_Operations	generic package	Provides a column matrix type and basic operations to go with it

3.6.8.2.9 PART DESIGN

3.6.8.2.9.1 VECTOR_OPERATIONS_UNCONSTRAINED (CATALOG #P168-0)

Taking the generic formal parameter "vector_elements", this generic package defines a vector as an unconstrained, one-dimensional array of these elements. It then defines operations on the vector. See decomposition section for operations provided.

No exceptions are raised by this package; however, exceptions are raised by routines in this package:

Name	Raised By	When/Why Raised
Dimension_Error	"+" "-" Dot_Product	Raised if the lengths of the two input vectors are not the same

3.6.8.2.9.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of requirements to this part:

Name	Requirements Allocation
"+"	R061
"-"	R062
Dot_Product	R063
Vector_Length	R104

3.6.8.2.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_Squared	floating point type	Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

DATA OBJECTS: None.

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared
SqRt	function	Square root function taking an object of type Vector_Elements_Squared and returning an object of type Vector_Elements

GLOBAL PARAMETERS: None.

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

EXCEPTIONS: None.

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Vectors	N/A	See decomposition section	Unconstrained array of Elements

3.6.8.2.9.1.3 LOCAL ENTITIES

None.

3.6.8.2.9.1.4 INTERRUPTS

None.

3.6.8.2.9.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Operations;
...
   type My_Indices is (a, b, c);
   ...
   type My_Elements         is new FLOAT;
   type My_Elements_Squared is new My_Elements;
   ...
   function "*" (Left  : My_Elements;
                 Right : My_Elements) return My_Elements_Squared;
   ...
   function SqRt (Input : Vector_Elements_Squared)
                 return Vector_Elements;
   ...
   package V_Opns is new
           General_Vector_Matrix_Operations.
              Vector_Operations_Unconstrained
                 (Vector_Elements         => My_Elements,
                  Vector_Elements_Squared => My_Elements_Squared,
                  Indices                 => My_Indices);
   use V_Opns;
   ...
   subtype My_Vectors is V_Opns.Vectors(My_Indices);
   ...
   Vector1 : My_Vectors;
   Vector2 : My_Vectors;
   ...
   begin
      ...
      Vector1 := Vector1 + Vector2;
      ...
```

3.6.8.2.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"+"	function	Adds two vectors with m elements c(i) = a(i) + b(i)
"-"	function	Subtracts two vectors with m c(i) = a(i) = b(i)
Dot_Product	function	Performs a dot product operation on two vectors with m elements c = a(i) * b(i)
Vector_Length	function	Calculates the length of a vector length = Sqrt(sum of a(i)**2)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"+"	P454-0
"-"	P455-0
Dot_Product	P456-0
Vector_Length	P457-0

3.6.8.2.9.1.8 PART DESIGN

None.

3.6.8.2.9.2 VECTOR_OPERATIONS_CONSTRAINED (CATALOG #P169-0)

Taking the generic formal parameter "vector_elements", this generic package defines a vector as an constrained, one-dimensional array of these elements. It then defines operations on the vector. See decomposition section for operations provided.

3.6.8.2.9.2.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
"+"	R061
"-"	R062
Dot_Product	R063
Vector_Length	R104

3.6.8.2.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_Squared	floating point type	Resulting type from the operation Vector_Elements * Vector_Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared
SqRt	function	Square root function taking an object of type Vector_Elements_Squared and returning an object of type Vector_Elements

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Vectors	N/A	See decomposition section	Constrained array of Elements

3.6.8.2.9.2.3 LOCAL ENTITIES

None.

3.6.8.2.9.2.4 INTERRUPTS

None.

3.6.8.2.9.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Operations;
...
   type My_Indices is (a, b, c);
   ...
   type My_Elements         is new FLOAT;
   type My_Elements_Squared is new My_Elements;
   ...
   function "*" (Left  : My_Elements;
                 Right : My_Elements) return My_Elements_Squared;
   ...
   function SqRt (Input : Vector_Elements_Squared)
                 return Vector_Elements;
   ...
   package V_Opns is new
           General_Vector_Matrix_Operations.
              Vector_Operations_Constrained
                 (Vector_Elements         => My_Elements,
                  Vector_Elements_Squared => My_Elements_Squared,
                  Indices                 => My_Indices);
   use V_Opns;
   ...
   Vector1 : VOpns.Vectors;
   Vector2 : VOpns.Vectors;
   ...
   begin
      ...
      Vector1 := Vector1 + Vector2;
      ...
```

3.6.8.2.9.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"+"	function	Adds two vectors with m elements c(i) = a(i) + b(i)
"-"	function	Subtracts two vectors with m c(i) = a(i) - b(i)
Dot_Product	function	Performs a dot product operation on two vectors with m elements c = a(i) * b(i)
Vector_Length	function	Calculates the length of a vector length = Sqrt(sum of a(i)**2)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"+"	P458-0
"-"	P459-0
Dot_Product	P460-0
Vector_Length	P461-0

3.6.8.2.9.2.8 PART DESIGN

None.

3.6.8.2.9.3 MATRIX_OPERATIONS_UNCONSTRAINED (CATALOG #P170-0)

Taking the generic formal parameter "elements", this generic package defines a matrix as an unconstrained, two-dimensional array of these elements. It then defines operations on the matrix. See the decomposition section for a description of the operations provided.

No exceptions are raised by this package; however, exceptions are raised by routines in this package:

Name	Raised By	When/Why Raised
Dimension_Error	"+" "-"	Raised if the sizes of the two input matrices are not the same
Dimension_Error	Set_to_Identity_Matrix	Raised if input matrix is not a square matrix
Dimension_Error	"*"	Raised if the lengths of the inner dimensions of the two input matrices are not the same

3.6.8.2.9.3.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
"+" (matrices + matrices)	R079
"-" (matrices - matrices)	R080
"+" (matrices + elements)	R075
"-" (matrices - elements)	R076
Set_to_Identity_Matrix	R155
Set_to_Zero_Matrix	R156
"*"	R077

3.6.8.2.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Matrices	N/A	See decomposition section	Unconstrained, two-dimensional array of Elements

3.6.8.2.9.3.3 LOCAL ENTITIES

None.

3.6.8.2.9.3.4 INTERRUPTS

None.

3.6.8.2.9.3.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices is new INTEGER 1..10;
   type My_Row_Indices is new INTEGER 1..10;
   ...
   type My_Elements    is new FLOAT;
   ...
   package M_Opns is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements,
               Row_Indices => My_Row_Indices);
   use M_Opns;
   ...
   subtype My_Matrices1 is M_Opns.Matrices(1..3, 4..6);
   subtype My_Matrices2 is M_Opns.Matrices(2..4, 5..7);
   subtype My_Matrices3 is M_Opns.Matrices(3..5, 6..8);
   ...
   Matrix1 : My_Matrices1;
   Matrix2 : My_Matrices2;
   Matrix3 : My_Matrices3;
   ...
   begin
      ...
      Matrix1 := Matrix2 + Matrix3;
```

3.6.8.2.9.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"+" (matrices + matrices)	function	Adds two m x n matrices c(i,j) = a(i,j) = b(i,j)
"-" (matrices - matrices)	function	Subtracts two m x n matrices c(i,j) = a(i,j) - b(i,j)
"+" (matrices + elements)	function	Adds a scalar value to each element of an m x n matrix c(i,j) = a(i,j) + b
"-" (matrices - elements)	function	Subtracts a scalar value from each element of an m x n matrix c(i,j) = a(i,j) - b
Set_to_Identity_Matrix	procedure	Initializes an m x m matrix to an m x m identity matrix
Set_to_Zero_Matrix	procedure	Sets all components of an m x m matrix to zero
"*"	function	Multiplies an m x n matrix by an n x p matrix, returning the resultant m x p matrix c(i,j) = a(i,k) * b(k,j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"+" (matrices + matrices)	P461-0
"-" (matrices - matrices)	P462-0
"+" (matrices + elements)	P463-0
"-" (matrices - elements)	P464-0
Set_to_Identity_Matrix	P465-0
Set_to_Zero_Matrix	P466-0
"*"	P467-0

3.6.8.2.9.3.8 PART DESIGN

None.

3.6.8.2.9.4 MATRIX_OPERATIONS_CONSTRAINED (CATALOG #P171-0)

Taking the generic formal parameter "elements", this generic package defines a matrix as a constrained, two-dimensional array of these elements. It then defines operations on the matrix. See the decomposition section for a description of the operations provided.

No exceptions are raised by this package; however, exceptions are raised by routines in this package:

Name	Raised By	When/Why Raised
Dimension_Error	Set_to_Identity_Matrix	Raised if input matrix is not a square matrix

3.6.8.2.9.4.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
"+" (matrices + matrices)	R079
"-" (matrices - matrices)	R080
"+" (matrices + elements)	R075
"-" (matrices - elements)	R076
Set_to_Identity_Matrix	R155
Set_to_Zero_Matrix	R156

3.6.8.2.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Matrices	N/A	See decomposition section	Constrained, two-dimensional array of Elements

3.6.8.2.9.4.3 LOCAL ENTITIES

None.

3.6.8.2.9.4.4 INTERRUPTS

None.

3.6.8.2.9.4.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices is new INTEGER 1..10;
   type My_Row_Indices is new INTEGER 1..10;
   ...
   type My_Elements    is new FLOAT;
   ...
   package M_Opns is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements,
               Row_Indices => My_Row_Indices);
   use M_Opns;
   ...
   Matrix1 : M_Opns.Matrices;
   Matrix2 : M_Opns.Matrices;
   Matrix3 : M_Opns.Matrices;
   ...
   begin
      ...
      Matrix1 := Matrix2 + Matrix3;
```

3.6.8.2.9.4.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.4.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"+" (matrices + matrices)	function	Adds two m x n matrices c(i,j) = a(i,j) + b(i,j)
"-" (matrices - matrices)	function	Subtracts two m x n matrices c(i,j) = a(i,j) - b(i,j)
"+" (matrices + elements)	function	Adds a scalar value to each element of an m x n matrix c(i,j) = a(i,j) + b
"-" (matrices - elements)	function	Subtracts a scalar value from each element of an m x n matrix c(i,j) = a(i,j) - b
Set_to_Identity_Matrix	procedure	Initializes an m x m matrix to an m x m identity matrix
Set_to_Zero_Matrix	procedure	Sets all components of an m x m matrix to zero

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"+" (matrices + matrices)	P468-0
"-" (matrices - matrices)	P469-0
"+" (matrices + elements)	P470-0
"-" (matrices - elements)	P471-0
Set_to_Identity_Matrix	P472-0
Set_to_Zero_Matrix	P473-0

3.6.8.2.9.4.8 PART DESIGN

None.

3.6.8.2.9.5 DYNAMICALLY_SPARSE_MATRIX_OPERATIONS_UNCONSTRAINED (CATALOG #P172-0)

This package defines a dynamically sparse matrix and operations on it. All elements of the matrix are stored, but most of the elements are expected to be 0. Which elements are zero does not have to remain the same. See decomposition section for the operations provided.

No exceptions are raised by this package. However, exceptions are raised by routines it contains:

Name	Raised By	When/Why Raised
Dimension_ Error	Set_to_Identity_Matrix Add_to_Identity Subtract_from_Identity	Raised if input matrices are not square matrices
Dimension_ Error	"+" "-"	Raised if both input matrices are not m x n matrices

3.6.8.2.9.5.1 REQUIREMENTS ALLOCATION

This part satisfies CAMP requirement R226.

3.6.8.2.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Matrices	N/A	See decomposition section	Unconstrained, two-dimensional array of Elements

3.6.8.2.9.5.3 LOCAL ENTITIES

None.

3.6.8.2.9.5.4 INTERRUPTS

None.

3.6.8.2.9.5.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices is new INTEGER 1..10;
   type My_Row_Indices is new INTEGER 1..10;
   ...
   type My_Elements    is new FLOAT;
   ...
   package Sparse_M_Opns is new
           General_Vector_Matrix_Algebra.
              Dynamically_Sparse_Matrix_Operations_Unconstrained
                 (Col_Indices => My_Col_Indices,
                  Elements    => My_Elements,
                  Row_Indices => My_Row_Indices);
   use Sparse_M_Opns;
   ...
   subtype My_Matrices1 is Sparse_M_Opns.Matrices(1..3, 4..6);
   subtype My_Matrices2 is Sparse_M_Opns.Matrices(2..4, 5..7);
   subtype My_Matrices3 is Sparse_M_Opns.Matrices(3..5, 6..8);
   ...
   Matrix1 : My_Matrices1;
   Matrix2 : My_Matrices2;
   Matrix3 : My_Matrices3;
   ...
   begin
      ...
      Matrix1 := Matrix2 + Matrix3;
```

3.6.8.2.9.5.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.5.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Set_to_Identity_Matrix	procedure	Sets a square matrix to an identity matrix
Set_to_Zero_Matrix	procedure	Sets each element of a square matrix to zero
Add_to_Identity	function	Adds a square input matrix to an identity matrix
Subtract_from_Identity	function	Subtracts a square input matrix from an identity matrix
"+"	function	Adds two m x n matrices c(i,j) = a(i,j) + b(i,j)
"-"	function	Subtracts two m x n matrices c(i,j) = a(i,j) - b(i,j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
Set_to_Identity_Matrix	P475-0
Set_to_Zero_Matrix	P476-0
Add_to_Identity	P477-0
Subtract_from_Identity	P478-0
"+"	P479-0
"-"	P480-0

3.6.8.2.9.5.8 PART DESIGN

None.

3.6.8.2.9.6 DYNAMICALLY_SPARSE_MATRIX_OPERATIONS_CONSTRAINED (CATALOG #P173-0)

This package defines a dynamically sparse matrix and operations on it. All elements of the matrix are stored, but most of the elements are expected to be 0. Which elements are zero does not have to remain the same. See decomposition section for the operations provided.

No exceptions are raised by this package. However, exceptions are raised by routines it contains:

Name	Raised By	When/Why Raised
Dimension_ Error	Set_to_Identity_Matrix Add_to_Identity Subtract_from_Identity	Raised if input matrices are not square matrices
Dimension_ Error	"+" "-"	Raised if both input matrices are not m x n matrices

ן)ע

3.6.8.2.9.6.1 REQUIREMENTS ALLOCATION

This part satisfies CAMP requirement R226.

3.6.8.2.9.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Matrices	N/A	See decomposition section	Cconstrained, two-dimensional array of Elements

3.6.8.2.9.6.3 LOCAL ENTITIES

None.

3.6.8.2.9.6.4 INTERRUPTS

None.

3.6.8.2.9.6.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices is new INTEGER 1..10;
   type My_Row_Indices is new INTEGER 1..10;
   ...
```

```
type My_Elements    is new FLOAT;
...
package Sparse_M_Opns is new
        General_Vector_Matrix_Algebra.
           Dynamically_Sparse_Matrix_Operations_Constrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements,
               Row_Indices => My_Row_Indices);
use Sparse_M_Opns;
...
subtype My_Matrices1 is Sparse_M_Opns.Matrices(1..3, 4..6);
subtype My_Matrices2 is Sparse_M_Opns.Matrices(2..4, 5..7);
subtype My_Matrices3 is Sparse_M_Opns.Matrices(3..5, 6..8);
...
Matrix1 : Sparse_M_Opns.Matrices;
Matrix2 : Sparse_M_Opns.Matrices;
Matrix3 : Sparse_M_Opns.Matrices;
...
begin
   ...
   Matrix1 := Matrix2 + Matrix3;
```

3.6.8.2.9.6.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.6.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Set_to_Identity_Matrix	procedure	Sets a square matrix to an identity matrix
Set_to_Zero_Matrix	procedure	Sets each element of a square matrix to zero
Add_to_Identity	function	Adds a square input matrix to an identity matrix
Subtract_from_Identity	function	Subtracts a square input matrix from an identity matrix
"+"	function	Adds two m x n matrices c(i,j) = a(i,j) + b(i,j)
"-"	function	Subtracts two m x n matrices c(i,j) = a(i,j) - b(i,j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
Set_to_Identity_Matrix	P482-0
Set_to_Zero_Matrix	P474-0
Add_to_Identity	P481-0
Subtract_from_Identity	P483-0
"+"	P484-0
"-"	P485-0

3.6.8.2.9.6.8 PART DESIGN

None.

3.6.8.2.9.7 SYMMETRIC_HALF_STORAGE_MATRIX_OPERATIONS (CATALOG #P174-0)

This package defines a symmetric half storage matrix and provides operations on it. For the operations provided, see the decomposition section. The bottom half the the matrix will be stored in row-major order.

The following table describes the exceptions raised by this package:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of Col_Indices and Row_Indices is not the same

3.6.8.2.9.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.6.8.2.9.7.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_Indices	discrete type	Used to dimension column slices
Row_Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored
Col_Slices	array	Data type defining a column slice of a matrix
Row_Slices	array	Data type defining a row slice of a matrix

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Matrices	N/A	See decomposition section	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following chart describes the data objects exported by this part:

Name	Type	Value	Description
Entry_Count	Positive	---	Number of stored values from the half-storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2

3.6.8.2.9.7.3 LOCAL ENTITIES

None.

3.6.8.2.9.7.4 INTERRUPTS

None.

3.6.8.2.9.7.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   ...
   type My_Elements         is new FLOAT;
   type My_Elements_Squared is new FLOAT;
   ...
   function "*" (Left  : My_Elements;
                 Right : My_Elements) return My_Elements_Squared;
   ...
   function Sqrt (Input : My_Elements_Squared) return My_Elements;
   ...
   package V_Opns_XYZ is new
           General_Vector_Matrix_Algebra.Vector_Operations
              (Vector_Elements         => My_Elements,
               Vector_Elements_Squared => My_Elements_Squared,
               Indices                 => My_Row_Indices);
   use V_Opns_XYZ;
   ...
   subtype Vectors_XYZ is V_Opns_XYZ.Vectors(My_Row_Indices);
   ...
   package V_Opns_ABC is new
           General_Vector_Matrix_Algebra.Vector_Operations
              (Vector_Elements         => My_Elements,
               Vector_Elements_Squared => My_Elements_Squared,
               Indices                 => My_Col_Indices);
   use V_Opns_ABC;
   ...
   subtype Vectors_ABC is V_Opns_ABC.Vectors(My_Col_Indices);
   ...
   package HStorage_M_Opns is new
           General_Vector_Matrix_Algebra.
              Symmetric_Half_Storage_Matrix_Operations
                 (Elements => My_Elements,
                  Col_Indices => My_Col_Indices,
                  Row_Indices => My_Row_Indices,
                  Col_Slices  => Vectors_XYZ,
                  Row_Slices  => Vectors_ABC);
   use HStorage_M_Opns;
   ...
   Half_Matrix : HStorage_M_Opns.Matrices;
   ...
   begin
      ...
      Half_Matrix := HStorage_M_Opns.Identity_Matrix;
      ...
```

3.6.8.2.9.7.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.7.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Initialize	procedure	Initializes matrix a row at a time
Identity_Matrix	function	Sets matrix to an identity matrix
Zero_Matrix	function	Sets each element of the matrix to zero
Change_Element	procedure	Changes a single element of the matrix
Retrieve_Element	function	Retrieves a single element of the matrix
Row_Slice	function	Retrieves one row from the matrix
Column_Slice	function	Retrieves one column from the matrix
Add_to_Identity	function	Adds input matrix to an identity matrix
Subtract_from_Identity	function	Subtracts input matrix from an identity matrix
"+"	function	Adds two half-storage matrices $c(i,j) = a(i,j) + b(i,j)$
"-"	function	Subtracts two half-storage matrices $c(i,j) = a(i,j) - b(i,j)$

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
Initialize	P486-0
Identity_Matrix	P487-0
Zero_Matrix	P488-0
Change_Element	P489-0
Retrieve_Element	P490-0
Row_Slice	P491-0
Column_Slice	P492-0
Add_to_Identity	P493-0
Subtract_from_Identity	P494-0
"+"	P495-0
"-"	P496-0

3.6.8.2.9.7.8 PART DESIGN

None.

3.6.8.2.9.8 SYMMETRIC_FULL_STORAGE_MATRIX_OPERATIONS_UNCONSTRAINED (CATALOG #P175-0)

This package defines a symmetric full storage matrix type and provides operations on it. For the operations provided, see the decomposition section. All elements of the matrix are stored, but operations on the matrices will take advantage of the fact that they are symmetric.

No exception are raised by this package. Exceptions are, however, raised by routines in this package.

Name	Raised By	When/Why Raised
Dimension_ Error	Set_to_Identity_Matrix Set_to_Zero_Matrix Add_to_Identity Subtract_from_Identity Change_Element	Raised if the input matrix is not a square matrix
Dimension_ Error	"+" "-"	Raised if both input matrices are not m x m matrices
Invalid_ Index	Change_Element	Raised if an attempt is made to change an element that is beyond the dimensions of the matrix

3.6.8.2.9.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.6.8.2.9.8.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following chart describes the exceptions exported by this part:

Name	Description
Invalid_Index	Indicates an attempt was made to access an element beyond the dimensions of the array

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Matrices	N/A	See decomposition	Unconstrained, two-dimensional array of Elements

3.6.8.2.9.8.3 LOCAL ENTITIES

None.

3.6.8.2.9.8.4 INTERRUPTS

None.

3.6.8.2.9.8.5 TIMING AND SEQUENCING

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices is new INTEGER 1..10;
   type My_Row_Indices is new INTEGER 1..10;
   ...
   type My_Elements    is new FLOAT;
   ...
   package Full_Storage_M_Opns is new
           General_Vector_Matrix_Algebra.
              Symmetric_Full_Storage_Matrix_Operations_Unconstrained
                 (Col_Indices => My_Col_Indices,
                  Elements    => My_Elements,
                  Row_Indices => My_Row_Indices);
   use Full_Storage_M_Opns;
   ...
   subtype My_Matrices1 is Full_Storage_M_Opns.Matrices(1..3, 4..6);
   subtype My_Matrices2 is Full_Storage_M_Opns.Matrices(2..4, 5..7);
   subtype My_Matrices3 is Full_Storage_M_Opns.Matrices(3..5, 6..8);
   ...
   Matrix1 : My_Matrices1;
   Matrix2 : My_Matrices2;
   Matrix3 : My_Matrices3;
   ...
   begin
```

```
...
Matrix1 := Matrix2 + Matrix3;
```

3.6.8.2.9.8.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.8.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Change_Element	procedure	Changes an element, along with its symmetric counterpart
Set_to_Identity_Matrix	procedure	Sets square matrix to an identity matrix
Set_to_Zero_Matrix	procedure	Sets each element of a square matrix to zero
Add_to_Identity	function	Adds square matrix to an identity matrix
Subtract_from_Identity	function	Subtracts square matrix from an identity matrix
"+"	function	Adds two square matrices c(i,j) = a(i,j) + b(i,j)
"-"	function	Subtracts two square matrices c(i,j) = a(i,j) - b(i,j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
Change_Element	P497-0
Set_to_Identity_Matrix	P498-0
Set_to_Zero_Matrix	P499-0
Add_to_Identity	P500-0
Subtract_from_Identity	P501-0
"+"	P502-0
"-"	P503-0

3.6.8.2.9.8.8 PART DESIGN

None.

3.6.8.2.9.9 SYMMETRIC_FULL_STORAGE_MATRIX_OPERATIONS_CONSTRAINED (CATALOG #P176-0)

This package defines a symmetric full storage matrix type and provides operations on it. For the operations provided, see the decomposition section. All elements of the matrix are stored, but operations on the matrices will take

advantage of the fact that they are symmetric.

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if an attempt is made to instantiate other than a square matrix

3.6.8.2.9.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.6.8.2.9.9.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Matrices	N/A	See decomposition	Constrained, two-dimensional array of Elements

3.6.8.2.9.9.3 LOCAL ENTITIES

None.

3.6.8.2.9.9.4 INTERRUPTS

None.

3.6.8.2.9.9.5 TIMING AND SEQUENCING

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices is new INTEGER 1..10;
   type My_Row_Indices is new INTEGER 1..10;
   ...
   type My_Elements    is new FLOAT;
   ...
   package Full_Storage_M_Opns is new
           General_Vector_Matrix_Algebra.
              Symmetric_Full_Storage_Matrix_Operations_Constrained
                 (Col_Indices => My_Col_Indices,
                  Elements    => My_Elements,
                  Row_Indices => My_Row_Indices);
   use Full_Storage_M_Opns;
   ...
   Matrix1 : Full_Storage_M_Opns.Matrices;
   Matrix2 : Full_Storage_M_Opns.Matrices;
   Matrix3 : Full_Storage_M_Opns.Matrices;
   ...
   begin
      ...
      Matrix1 := Matrix2 + Matrix3;
```

3.6.8.2.9.9.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.9.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Change_Element	procedure	Changes an element, along with its symmetric counterpart
Set_to_Identity_Matrix	procedure	Sets square matrix to an identity matrix
Set_to_Zero_Matrix	procedure	Sets each element of a square matrix to zero
Add_to_Identity	function	Adds square matrix to an identity matrix
Subtract_from_Identity	function	Subtracts square matrix from an identity matrix
"+"	function	Adds two square matrices c(i,j) = a(i,j) + b(i,j)
"-"	function	Subtracts two square matrices c(i,j) = a(i,j) - b(i,j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
Change_Element	P504-0
Set_to_Identity_Matrix	P505-0
Set_to_Zero_Matrix	P506-0
Add_to_Identity	P507-0
Subtract_from_Identity	P508-0
"+"	P509-0
"-"	P510-0

3.6.8.2.9.9.8 PART DESIGN

None.

3.6.8.2.9.10 DIAGONAL_MATRIX_OPERATIONS (CATALOG #P177-0)

This package defines a diagonal matrix where the only non-zero elements occur on the diagonal. It then provides operations on that type. For the operations provided, see the decomposition section.

Name	Raised By	When/Why Raised
Dimension_Error	this package	Raised if the lengths of column and row indices are not the same
Invalid_Index	Change_Element Retrieve_Element	Raised if element requested does not fall on the diagonal

3.6.8.2.9.10.1 REQUIREMENTS ALLOCATION

This part meets CAMP require R212.

3.6.8.2.9.10.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays
Col_Slices	array	One-dimensional array of column Elements
Row_Slices	array	One-dimensional array of row Elements

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following table describes the exceptions raised by this part:

Name	Description
Invalid_Index	Indicates an attempt was made to access an element not on the diagonal

Data types:

The following chart describes the data types exported by this part:

Name	Range	Operators	Description
Diagonal_ Range	1.. Entry_ Count	N/A	Used to dimension diagonal_ matrices
Diagonal_ Matrices	N/A	See decomposition section	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

The following table describes the data objects exported by this part:

Name	Type	Description
Entry_Count	Positive	Number of diagonal elements in the array

3.6.8.2.9.10.3 LOCAL ENTITIES

None.

3.6.8.2.9.10.4 INTERRUPTS

None.

3.6.8.2.9.10.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   ...
   type My_Elements         is new FLOAT;
   type My_Elements_Squared is new FLOAT;
   ...
   function "*" (Left  : My_Elements;
                 Right : My_Elements) return My_Elements_Squared;
   ...
   function Sqrt (Input : My_Elements_Squared) return My_Elements;
   ...
   package V_Opns_XYZ is new
           General_Vector_Matrix_Algebra.Vector_Operations
              (Vector_Elements         => My_Elements,
               Vector_Elements_Squared => My_Elements_Squared,
               Indices                 => My_Row_Indices);
   use V_Opns_XYZ;
   ...
   subtype Vectors_XYZ is V_Opns_XYZ.Vectors(My_Row_Indices);
   ...
   package V_Opns_ABC is new
           General_Vector_Matrix_Algebra.Vector_Operations
              (Vector_Elements         => My_Elements,
               Vector_Elements_Squared => My_Elements_Squared,
               Indices                 => My_Col_Indices);
   use V_Opns_ABC;
   ...
   subtype Vectors_ABC is V_Opns_ABC.Vectors(My_Col_Indices);
   ...
   package Diagonal_M_Opns is new
           General_Vector_Matrix_Algebra.
              Diagonal_Matrix_Operations
```

```
                 (Elements     => My_Elements,
                  Col_Indices  => My_Col_Indices,
                  Row_Indices  => My_Row_Indices,
                  Col_Slices   => Vectors_XYZ,
                  Row_Slices   => Vectors_ABC);
   use Diagonal_M_Opns;
   ...
   Diag_Matrix : Diagonal_M_Opns.Diagonal_Matrices;
   ...
   begin
      ...
      Diag_Matrix := Diagonal_M_Opns.Identity_Matrix;
      ...
```

3.6.8.2.9.10.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.10.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Identity_Matrix	function	Returns an identity matrix
Zero_Matrix	function	Returns a zero matrix
Change_Element	procedure	Changes a single element of a diagonal matrix
Retrieve_Element	function	Retrieves a single element from a diagonal matrix
Row_Slice	function	Retrieves a row from a diagonal matrix
Column_Slice	function	Retrieves a column from a diagonal matrix
Add_to_Identity	function	Adds an input diagonal matrix to an identity matrix
Subtract_from_Identity	function	Subtracts an input diagonal matrix from an identity matrix
"+"	function	Adds two diagonal matrices c(i,j) = a(i,j) + b(i,j)
"-"	function	Subtracts two diagonal matrices c(i,j) = a(i,j) - b(i,j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
Identity_Matrix	P511-0
Zero_Matrix	P512-0
Change_Element	P513-0
Retrieve_Element	P514-0
Row_Slice	P515-0
Column_Slice	P516-0
Add_to_Identity	P517-0
Subtract_from_Identity	P518-0
"+"	P519-0
"-"	P520-0

3.6.8.2.9.10.8 PART DESIGN

None.

3.6.8.2.9.11 VECTOR_SCALAR_OPERATIONS_UNCONSTRAINED (CATALOG #P178-0)

This package provides the functions to allow the user to multiply or divide each element of a vector by a scalar.

3.6.8.2.9.11.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
"*"	R065
"/"	R066

3.6.8.2.9.11.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices1	discrete	Used to dimension Vectors1
Indices2	discrete	Used to dimension Vectors2
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Used to define the operation Elements1 := Elements2 * Scalars
"/"	function	Used to define the operation Elements2 := Elements1 / Scalars

3.6.8.2.9.11.3 LOCAL ENTITIES

None.

3.6.8.2.9.11.4 INTERRUPTS

None.

3.6.8.2.9.11.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Indices is (a, b, c);
   ...
   type My_Elements1           is new FLOAT;
   type My_Elements1_Squared is new My_Elements1;
   type My_Elements2           is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type Scalars                is new FLOAT;
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements1) return My_Elements1_Squared;
   ...
```

```
function "*" (Left  : My_Elements2,
              Right : My_Elements2) return My_Elements2_Squared;
...
function "*" (Left  : My_Elements2,
              Right : Scalars) return My_Elements1;
...
function "/" (Left  : My_Elements1,
              Right : Scalars) return My_Elements2;
...
function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
...
package V_Opns1 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements         => My_Elements1,
            Vector_Elements_Squared => My_Elements1_Squared,
            Indices                 => My_Indices);
use V_Opns1;
...
subtype Vectors1 is V_Opns1.Vectors(My_Indices);
...
package V_Opns2 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements         => My_Elements2,
            Vector_Elements_Squared => My_Elements2_Squared,
            Indices                 => My_Indices);
use V_Opns2;
...
subtype Vectors2 is V_Opns2.Vectors(My_Indices);
...
package V_S_Opns is new
        General_Vector_Matrix_Algebra.
           Vector_Scalar_Operations_Unconstrained
              (Elements1 => My_Elements1,
               Elements2 => My_Elements2,
               Indices1  => My_Indices,
               Indices2  => My_Indices,
               Vectors1  => Vectors1,
               Vectors2  => Vectors2,
               Scalars   => Scalars);
use V_S_Opns;
...
Vector1 : Vectors1;
Vector2 : Vectors2;
Gain    : Scalars;
...
begin
   ...
   Vector1 := Vector2 * Gain;
   ...
```

3.6.8.2.9.11.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.11.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies each element of a vector by a scalar value c(i) = a(i) * b
"/"	function	Divides each element of a vector by a scalar value c(i) = a(i) / b

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P521-0
"/"	P522-0

3.6.8.2.9.11.8 PART DESIGN

None.

3.6.8.2.9.12 VECTOR_SCALAR_OPERATIONS_CONSTRAINED (CATALOG #P179-0)

This package provides the functions to allow the user to multiply or divide each element of a vector by a scalar.

3.6.8.2.9.12.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
"*"	R065
"/"	R066

3.6.8.2.9.12.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices	discrete	Used to dimension Vectorsx
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Used to define the operation Elements1 := Elements2 * Scalars
"/"	function	Used to define the operation Elements2 := Elements1 / Scalars

3.6.8.2.9.12.3 LOCAL ENTITIES

None.

3.6.8.2.9.12.4 INTERRUPTS

None.

3.6.8.2.9.12.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Indices is (a, b, c);
   ...
   type My_Elements1         is new FLOAT;
   type My_Elements1_Squared is new My_Elements1;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type Scalars              is new FLOAT;
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements1) return My_Elements1_Squared;
```

```
...
function "*" (Left  : My_Elements2,
              Right : My_Elements2) return My_Elements2_Squared;
...
function "*" (Left  : My_Elements2,
              Right : Scalars) return My_Elements1;
...
function "/" (Left  : My_Elements1,
              Right : Scalars) return My_Elements2;
...
function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
...
package V_Opns1 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Constrained
           (Vector_Elements         => My_Elements1,
            Vector_Elements_Squared => My_Elements1_Squared,
            Indices                 => My_Indices);
use V_Opns1;
...
subtype Vectors1 is V_Opns1.Vectors;
...
package V_Opns2 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Constrained
           (Vector_Elements         => My_Elements2,
            Vector_Elements_Squared => My_Elements2_Squared,
            Indices                 => My_Indices);
use V_Opns2;
...
subtype Vectors2 is V_Opns2.Vectors;
...
package V_S_Opns is new
        General_Vector_Matrix_Algebra.Vector_Scalar_Operations
           (Elements1 => My_Elements1,
            Elements2 => My_Elements2,
            Indices   => My_Indices,
            Vectors1  => Vectors1,
            Vectors2  => Vectors2,
            Scalars   => Scalars);
use V_S_Opns;
...
Vector1 : Vectors1;
Vector2 : Vectors2;
Gain    : Scalars;
...
begin
   ...
   Vector1 := Vector2 * Gain;
   ...
```

3.6.8.2.9.12.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.12.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies each element of a vector by a scalar value c(i) = a(i) * b
"/"	function	Divides each element of a vector by a scalar value c(i) = a(i) / b

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P523-0
"/"	P524-0

3.6.8.2.9.12.8 PART DESIGN

None.

3.6.8.2.9.13 MATRIX_SCALAR_OPERATIONS_UNCONSTRAINED (CATALOG #P180-0)

This package provides a set of functions which will scale a matrix by multiplying or dividing each element of the matrix by a scale factor.

3.6.8.2.9.13.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
"*"	R073
"/"	R074

3.6.8.2.9.13.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col_Indices1	discrete type	Used to dimension second dimension of Matrices1
Row_Indices1	discrete type	Used to dimension first dimension of Matrices1
Col_Indices2	discrete type	Used to dimension second dimension of Matrices2
Row_Indices2	discrete type	Used to dimension first dimension of Matrices2
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function to define the operation Elements1 * Scalars := Elements2
"/"	function	Function to define the operation Elements2 / Scalars := Elements1

3.6.8.2.9.13.3 LOCAL ENTITIES

None.

3.6.8.2.9.13.4 INTERRUPTS

None.

3.6.8.2.9.13.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1 is new FLOAT;
```

```
type My_Elements2 is new FLOAT;
type Scalars      is new FLOAT;
...
type My_Col_Indices is (a, b, c);
type My_Row_Indices is (x, y, z);
...
function "*" (Left  : My_Elements1;
              Right : Scalars) return My_Elements2;
...
function "/" (Left  : My_Elements2;
              Right : Scalars) return My_Elements1;
...
package M_Opns1 is new
        General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
           (Col_Indices => My_Col_Indices,
            Elements    => My_Elements1,
            Row_Indices => My_Row_Indices);
use M_Opns1;
...
subtype Matrices1 is M_Opns1.Matrices(My_Row_Indices, My_Col_Indices);
...
package M_Opns2 is new
        General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
           (Col_Indices => My_Col_Indices,
            Elements    => My_Elements2,
            Row_Indices => My_Row_Indices);
use M_Opns2;
...
subtype Matrices2 is M_Opns2.Matrices(My_Row_Indices, My_Col_Indices);
...
package M_S_Opns is new
        General_Vector_Matrix_Algebra.
           Matrix_Scalar_Operations_Unconstrained
              (Elements1    => My_Elements1,
               Elements2    => My_Elements2,
               Scalars      => Scalars,
               Col_Indices1 => My_Col_Indices,
               Col_Indices2 => My_Col_Indices,
               Row_Indices1 => My_Row_Indices,
               Row_Indices2 => My_Row_Indices,
               Matrices1    => Matrices1,
               Matrices2    => Matrices2);
use M_S_Opns;
...
Matrix1 : Matrices1;
Matrix2 : Matrices2;
Gain    : Scalars;
...
begin
   ...
   Matrix2 := Matrix1 * Gain;
```

3.6.8.2.9.13.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.13.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies each element of an m x n matrix by a scalar value c(i) = a(i) * b
"/"	function	Divides each element of an m x n matrix by a scalar value c(i) = a(i) / b

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P525-0
"/"	P526-0

3.6.8.2.9.13.8 PART DESIGN

None.

3.6.8.2.9.14 MATRIX_SCALAR_OPERATIONS_CONSTRAINED (CATALOG #P181-0)

This package provides a set of functions which will scale a matrix by multiplying or dividing each element of the matrix by a scale factor.

3.6.8.2.9.14.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
"*"	R073
"/"	R074

3.6.8.2.9.14.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col_Indices	discrete type	Used to dimension second dimension of Matricesx
Row_Indices	discrete type	Used to dimension first dimension of Matricesx
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function to define the operation Elements1 * Scalars := Elements2
"/"	function	Function to define the operation Elements2 / Scalars := Elements1

3.6.8.2.9.14.3 LOCAL ENTITIES

None.

3.6.8.2.9.14.4 INTERRUPTS

None.

3.6.8.2.9.14.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
```

```
...
   type My_Elements1 is new FLOAT;
   type My_Elements2 is new FLOAT;
   type Scalars      is new FLOAT;
   ...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   ...
   function "*" (Left  : My_Elements1;
                 Right : Scalars) return My_Elements2;
   ...
   function "/" (Left  : My_Elements2;
                 Right : Scalars) return My_Elements1;
   ...
   package M_Opns1 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements1,
               Row_Indices => My_Row_Indices);
   use M_Opns1;
   ...
   subtype Matrices1 is M_Opns1.Matrices;
    ...
   package M_Opns2 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements2,
               Row_Indices => My_Row_Indices);
   use M_Opns2;
   ...
   subtype Matrices2 is M_Opns2.Matrices;
   ...
   package M_S_Opns is new
           General_Vector_Matrix_Algebra.
              Matrix_Scalar_Operations_Constrained
                 (Elements1     => My_Elements1,
                  Elements2     => My_Elements2,
                  Scalars       => Scalars,
                  Col_Indices   => My_Col_Indices,
                  Row_Indices   => My_Row_Indices,
                  Matrices1     => Matrices1,
                  Matrices2     => Matrices2);
   use M_S_Opns;
   ...
   Matrix1  : Matrices1;
   Matrix2  : Matrices2;
   Gain     : Scalars;
   ...
   begin
     ...
      Matrix2 := Matrix1 * Gain;
```

3.6.8.2.9.14.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.14.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies each element of an m x n matrix by a scalar value c(i) = a(i) * b
"/"	function	Divides each element of an m x n matrix by a scalar value c(i) = a(i) / b

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P527-0
"/"	P528-0

3.6.8.2.9.14.8 PART DESIGN

None.

3.6.8.2.9.15 DIAGONAL_MATRIX_SCALAR_OPERATIONS (CATALOG #P182-0)

This package provides the functions to allow the user to multiply or divide each element of a diagonal matrix by a scalar.

This package has been designed to be instantiated by the diagonal matrix type exported by the Diagonal_Matrix_Operations package. However, a similarly defined diagonal matrix, defined by the user, may also be used to instantiated this package.

The following exceptions are raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the two imported vector types are not of the same length

3.6.8.2.9.15.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.6.8.2.9.15.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements1	floating point type	Type of elements in Diagonal_Matrices1
Elements2	floating point type	Type of elements in Diagonal_Matrices2
Scalars	floating point type	Data type of scale factor
Diagonal_Range1	integer type	Used to dimension Diagonal_Matrices1
Diagonal_Range2	integer type	Used to dimension Diagonal_Matrices2
Diagonal_Matrices1	array	An array of Elements1
Diagonal_Matrices2	array	An array of Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Elements1 * Scalars = Elements2
"/"	function	Division operator defining the operation: Elements2 / Scalars = Elements1

3.6.8.2.9.15.3 LOCAL ENTITIES

None.

3.6.8.2.9.15.4 INTERRUPTS

None.

3.6.8.2.9.15.5 TIMING AND SEQUENCING

The following shows a sample usage of thi part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   ...
   type My_Elements1         is new FLOAT;
   type My_Elements1_Squared is new FLOAT;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new FLOAT;
   type My_Elements3         is new FLOAT;
   ...
   function "*" (Left  : My_Elements1;
                 Right : My_Elements1) return My_Elements1_Squared;
   ...
   function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
   ...
   function "*" (Left  : My_Elements2;
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
   ...
   package V_Opns_XYZ1 is new
           General_Vector_Matrix_Algebra.Vector_Operations
              (Vector_Elements         => My_Elements1,
               Vector_Elements_Squared => My_Elements1_Squared,
               Indices                 => My_Row_Indices);
   use V_Opns_XYZ1;
   ...
   subtype Vectors_XYZ1 is V_Opns_XYZ1.Vectors(My_Row_Indices);
   ...
   package V_Opns_XYZ2 is new
           General_Vector_Matrix_Algebra.Vector_Operations
              (Vector_Elements         => My_Elements2,
               Vector_Elements_Squared => My_Elements2_Squared,
               Indices                 => My_Row_Indices);
   use V_Opns_XYZ2;
   ...
   subtype Vectors_XYZ2 is V_Opns_XYZ2.Vectors(My_Row_Indices);
   ...
   package V_Opns_ABC1 is new
           General_Vector_Matrix_Algebra.Vector_Operations
              (Vector_Elements         => My_Elements1,
               Vector_Elements_Squared => My_Elements1_Squared,
               Indices                 => My_Col_Indices);
   use V_Opns_ABC1;
   ...
   subtype Vectors_ABC1 is V_Opns_ABC1.Vectors(My_Col_Indices);
   ...
   package V_Opns_ABC2 is new
           General_Vector_Matrix_Algebra.Vector_Operations
              (Vector_Elements         => My_Elements2,
               Vector_Elements_Squared => My_Elements2_Squared,
               Indices                 => My_Col_Indices);
```

```
use V_Opns_ABC2;
...
subtype Vectors_ABC2 is V_Opns_ABC2.Vectors(My_Col_Indices);
...
package Diagonal_M_Opns1 is new
        General_Vector_Matrix_Algebra.
           Diagonal_Matrix_Operations
              (Elements        => My_Elements1,
               Col_Indices     => My_Col_Indices,
               Row_Indices     => My_Row_Indices,
               Col_Slices      => Vectors_XYZ1,
               Row_Slices      => Vectors_ABC1);
use Diagonal_M_Opns1;
...
package Diagonal_M_Opns2 is new
        General_Vector_Matrix_Algebra.
           Diagonal_Matrix_Operations
              (Elements        => My_Elements2,
               Col_Indices     => My_Col_Indices,
               Row_Indices     => My_Row_Indices,
               Col_Slices      => Vectors_XYZ2,
               Row_Slices      => Vectors_ABC2);
use Diagonal_M_Opns2;
...
package D_S_Opns is new
        General_Vector_Matrix_Algebra.
           Diagonal_Matrix_Scalar_Operations
              (Elements1            => My_Elements1,
               Elements2            => My_Elements2,
               Scalars              => My_Elements3,
               Diagonal_Range1      => Diagonal_M_Opns1.
                                          Diagonal_Range,
               Diagonal_Range2      => Diagonal_M_Opns2.
                                          Diagonal_Range,
               Diagonal_Matrices1 => Diagonal_M_Opns1.
                                          Diagonal_Matrices,
               Diagonal_Matrices2 => Diagonal_M_Opns2.
                                          Diagonal_Matrices);
use D_S_Opns;
...
Diag_Matrix1 : Diagonal_M_Opns1.Diagonal_Matrices;
Diag_Matrix2 : Diagonal_M_Opns2.Diagonal_Matrices;
Scale_Factor : My_Elements3;
...
begin
   ...
   Diag_Matrix2 := Diag_Matrix1 * Scale_Factor;
```

3.6.8.2.9.15.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.15.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies each element of a diagonal matrix by a scalar value a(i) = b(i) * c
"/"	function	Divides each element of a diagonal matrix by a scalar value a(i) = b(i) / c

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P529-0
"/"	P530-0

3.6.8.2.9.15.8 PART DESIGN

None.

3.6.8.2.9.16 MATRIX_VECTOR_MULTIPLY_UNRESTRICTED (CATALOG #P183-0)

This package contains a function which multiplies an m x n matrix by an n x 1 vector producing an m x 1 vector. If the length of the second dimension of the matrix is not the same as the length of the input vector, a DIMENSION_ERROR exception is raised. None of the ranges need to be the same.

The function in this package can be made to handle sparse matrices and/or vectors by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

The following exceptions are raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the length of the second dimension of the input matrix is not the same as the length of the vector

3.6.8.2.9.16.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R069.

3.6.8.2.9.16.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Col_Indices	discrete type	Used to dimension second dimension of input matrix
Row_Indices	discrete type	Used to dimension first dimension of input matrix
Input_Vector_Indices	discrete	Used to dimension input vector
Output_Vector_Indices	discrete	Used to dimension output vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Matrix_Elements * Input_Vector_Elements := Output_Vector_Elements
"+"	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements

3.6.8.2.9.16.3 LOCAL ENTITIES

None.

3.6.8.2.9.16.4 INTERRUPTS

None.

3.6.8.2.9.16.5 TIMING AND SEQUENCING

The following shows a sample usage of this part where neither the input matrix or vector are sparse:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   type My_Elements3_Squared is new My_Elements3;
   ...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   type My_Indices     is (i, j, k);
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;

   function "*" (Left  : My_Elements2,
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function "*" (Left  : My_Elements3,
                 Right : My_Elements3) return My_Elements3_Squared;
   ...
   function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
   function Sqrt (Input : My_Elements3_Squared) return My_Elements3;
   ...
   package M_Opns1 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements1,
               Row_Indices => My_Row_Indices);
   use M_Opns1;
   ...
   subtype Matrices1 is M_Opns1.Matrices(My_Row_Indices, My_Col_Indices);
   ...
   package V_Opns2 is new
           General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
              (Vector_Elements         => My_Elements2,
               Vector_Elements_Squared => My_Elements2_Squared,
               Indices                 => My_Indices);
   use V_Opns2;
   ...
   subtype Vectors2 is V_Opns2.Vectors(My_Indices);
   ...
   package V_Opns3 is new
           General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
              (Vector_Elements         => My_Elements3,
               Vector_Elements_Squared => My_Elements3_Squared,
               Indices                 => My_Indices);
```

```
   use V_Opns3;
   ...
   subtype Vectors3 is V_Opns3.Vectors(My_Indices);
   ...
   function "*" is new
            General_Vector_Matrix_Algebra.
               Matrix_Vector_Multiply_Unrestricted
                  (Matrix_Elements        => My_Elements1,
                   Input_Vector_Elements  => My_Elements2,
                   Output_Vector_Elements => My_Elements3,
                   Col_Indices            => My_Col_Indices,
                   Input_Vector_Indices   => My_Indices,
                   Output_Vector_Indices  => My_Indices,
                   Row_Indices            => My_Row_Indices,
                   Input_Matrices         => Matrices1,
                   Input_Vectors          => Vectors2,
                   Output_Vectors         => Vectors3);
   ...
   Matrix1 : Matrices1;
   Vector2 : Vectors2;
   Vector3 : Vectors3;
   ...
   begin
   ...
      Vector3 := Matrix1 * Vector2;
```

The following shows a sample usage of this part where the input matrix is sparse:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   type My_Elements3_Squared is new My_Elements3;
   ...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   type My_Indices     is (i, j, k);
   ...
   function Sparse_Left_Multiply
             (Left  : My_Elements1;
              Right : My_Elements2) return My_Elements3;
   ...
   function Sparse_Left_Add
             (Left  : My_Elements3;
              Right : My_Elements3) return My_Elements3;
   ...
   function "*" (Left  : My_Elements2,
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function "*" (Left  : My_Elements3,
                 Right : My_Elements3) return My_Elements3_Squared;
   ...
   function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
   function Sqrt (Input : My_Elements3_Squared) return My_Elements3;
```

```
...
package Sparse_M_Opns1 is new
        General_Vector_Matrix_Algebra.
        Dynamically_Sparse_Matrix_Operations_Unconstrained
           (Col_Indices => My_Col_Indices,
            Elements    => My_Elements1,
            Row_Indices => My_Row_Indices);
use Sparse_M_Opns1;
...
subtype Sp_Matrices1 is Sparse_M_Opns1.
                        Matrices(My_Row_Indices, My_Col_Indices);
...
package V_Opns2 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements         => My_Elements2,
            Vector_Elements_Squared => My_Elements2_Squared,
            Indices                 => My_Indices);
use V_Opns2;
...
subtype Vectors2 is V_Opns2.Vectors(My_Indices);
...
package V_Opns3 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements         => My_Elements3,
            Vector_Elements_Squared => My_Elements3_Squared,
            Indices                 => My_Indices);
use V_Opns3;
...
subtype Vectors3 is V_Opns3.Vectors(My_Indices);
...
package M_V_Mult is new
        General_Vector_Matrix_Algebra.
           Matrix_Vector_Multiply_Unrestricted
              (Matrix_Elements         => My_Elements1,
               Input_Vector_Elements   => My_Elements2,
               Output_Vector_Elements  => My_Elements3,
               Col_Indices             => My_Col_Indices,
               Input_Vector_Indices    => My_Indices,
               Output_Vector_Indices   => My_Indices,
               Row_Indices             => My_Row_Indices,
               Input_Matrices          => Sp_Matrices1,
               Input_Vectors           => Vectors2,
               Output_Vectors          => Vectors3,
               "*"                     => Sparse_Left_Multiply,
               "+"                     => Sparse_Left_Add);
use M_V_Mult;
...
Sp_Matrix1 : Sp_Matrices1;
Vector2    : Vectors2;
Vector3    : Vectors3;
...
function Sparse_Left_Multiply
            (Left  : My_Elements1;
             Right : My_Elements2) return My_Elements3 is
   Answer : My_Elements3;
begin
   if Left = 0.0 then
```

```
         Answer := My_Elements3(Right);
      else
         Answer := My_Elements3(Left * My_Elements1(Right));
      end if;
      return Answer;
   end Sparse_Left_Multiply;
   ...
   function Sparse_Left_Add
               (Left  : My_Elements3;
                Right : My_Elements3) return My_Elements3 is
      Answer : My_Elements3;
   begin
      if Left = 0.0 then
         Answer := Right;
      else
         Answer := Left + Right;
      end if;
      return Answer;
   end Sparse_Left_Add;
   ...
   begin
   ...
      Vector3 := Sp_Matrix1 * Vector2;
```

3.6.8.2.9.16.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.16.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies an m x n matrix by an n x 1 vector, returning the resultant m x 1 vector c(i) := a(i,j) * b(j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P531-0

3.6.8.2.9.16.8 PART DESIGN

None.

3.6.8.2.9.17 MATRIX_VECTOR_MULTIPLY_RESTRICTED (CATALOG #P184-0)

This function multiplies an m x n matrix by an n x 1 vector producing an m x 1 vector.

The function can be made to handle sparse matrices and/or vectors by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

3.6.8.2.9.17.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R069.

3.6.8.2.9.17.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Indices1	discrete type	Used to dimension first dimension of input matrix and to dimension the output vector
Indices2	discrete type	Used to dimension second dimension of input matrix and to dimension the input vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Matrix_Elements * Input_Vector_Elements := Output_Vector_Elements
"+"	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Input_Matrices	In	M x N input matrix
Vector	Input_Vectors	In	N-element vector

3.6.8.2.9.17.3 INTERRUPTS

None.

3.6.8.2.9.17.4 TIMING AND SEQUENCING

The following shows a sample usage of this part where neither the input matrix or vector are sparse:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   type My_Elements3_Squared is new My_Elements3;
   ...
   type My_Indices1 is (x, y, z);
   type My_Indices2 is (a, b, c);
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;

   function "*" (Left  : My_Elements2,
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function "*" (Left  : My_Elements3,
                 Right : My_Elements3) return My_Elements3_Squared;
   ...
   function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
   function Sqrt (Input : My_Elements3_Squared) return My_Elements3;
   ...
```

```
    package M_Opns1 is new
            General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
               (Col_Indices => My_Indices2,
                Elements    => My_Elements1,
                Row_Indices => My_Indices1);
    use M_Opns1;
    ...
    subtype Matrices1 is M_Opns1.Matrices;
    ...
    package V_Opns2 is new
            General_Vector_Matrix_Algebra.Vector_Operations_Constrained
               (Vector_Elements         => My_Elements2,
                Vector_Elements_Squared => My_Elements2_Squared,
                Indices                 => My_Indices1);
    use V_Opns2;
    ...
    subtype Vectors2 is V_Opns2.Vectors;
    ...
    package V_Opns3 is new
            General_Vector_Matrix_Algebra.Vector_Operations_Constrained
               (Vector_Elements         => My_Elements3,
                Vector_Elements_Squared => My_Elements3_Squared,
                Indices                 => My_Indices2);
    use V_Opns3;
    ...
    subtype Vectors3 is V_Opns3.Vectors;
    ...
    function "*" is new
             General_Vector_Matrix_Algebra.
                Matrix_Vector_Multiply_Constrained
                   (Matrix_Elements        => My_Elements1,
                    Input_Vector_Elements  => My_Elements2,
                    Output_Vector_Elements => My_Elements3,
                    Indices1               => My_Indices1,
                    Indices2               => My_Indices2,
                    Input_Matrices         => Matrices1,
                    Input_Vectors          => Vectors2,
                    Output_Vectors         => Vectors3);
    ...
    Matrix1 : Matrices1;
    Vector2 : Vectors2;
    Vector3 : Vectors3;
    ...
    begin
    ...
       Vector3 := Matrix1 * Vector2;
```

The following shows a sample usage of this part where the input matrix is sparse:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   type My_Elements3_Squared is new My_Elements3;
```

```
...
type My_Indices1 is (x, y, z);
type My_Indices2 is (a, b, c);
...
function Sparse_Left_Multiply
            (Left  : My_Elements1;
             Right : My_Elements2) return My_Elements3;
...
function Sparse_Left_Add
            (Left  : My_Elements3;
             Right : My_Elements3) return My_Elements3;
...
function "*" (Left  : My_Elements2,
              Right : My_Elements2) return My_Elements2_Squared;
...
function "*" (Left  : My_Elements3,
              Right : My_Elements3) return My_Elements3_Squared;
...
function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
function Sqrt (Input : My_Elements3_Squared) return My_Elements3;
...
package Sparse_M_Opns1 is new
        General_Vector_Matrix_Algebra.
        Dynamically_Sparse_Matrix_Operations_Constrained
           (Col_Indices => My_Indices2,
            Elements    => My_Elements1,
            Row_Indices => My_Indices1);
use Sparse_M_Opns1;
...
subtype Sp_Matrices1 is Sparse_M_Opns1.Matrices;
...
package V_Opns2 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Constrained
           (Vector_Elements         => My_Elements2,
            Vector_Elements_Squared => My_Elements2_Squared,
            Indices                 => My_Indices1);
use V_Opns2;
...
subtype Vectors2 is V_Opns2.Vectors;
...
package V_Opns3 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Constrained
           (Vector_Elements         => My_Elements3,
            Vector_Elements_Squared => My_Elements3_Squared,
            Indices                 => My_Indices2);
use V_Opns3;
...
subtype Vectors3 is V_Opns3.Vectors(My_Indices);
...
package M_V_Mult is new
        General_Vector_Matrix_Algebra.
           Matrix_Vector_Multiply_Restricted
              (Matrix_Elements        => My_Elements1,
               Input_Vector_Elements  => My_Elements2,
               Output_Vector_Elements => My_Elements3,
               Indices1               => My_Indices1,
               Indices2               => My_Indices2,
```

```
                  Input_Matrices          => Sp_Matrices1,
                  Input_Vectors           => Vectors2,
                  Output_Vectors          => Vectors3,
                  "*"                     => Sparse_Left_Multiply,
                  "+"                     => Sparse_Left_Add);
   use M_V_Mult;
   ...
   Sp_Matrix1 : Sp_Matrices1;
   Vector2    : Vectors2;
   Vector3    : Vectors3;
   ...
   function Sparse_Left_Multiply
               (Left  : My_Elements1;
                Right : My_Elements2) return My_Elements3 is
      Answer : My_Elements3;
   begin
      if Left = 0.0 then
         Answer := My_Elements3(Right);
      else
         Answer := My_Elements3(Left * My_Elements1(Right));
      end if;
      return Answer;
   end Sparse_Left_Multiply;
   ...
   function Sparse_Left_Add
               (Left  : My_Elements3;
                Right : My_Elements3) return My_Elements3 is
      Answer : My_Elements3;
   begin
      if Left = 0.0 then
         Answer := Right;
      else
         Answer := Left + Right;
      end if;
      return Answer;
   end Sparse_Left_Add;
   ...
   begin
   ...
      Vector3 := Sp_Matrix1 * Vector2;
```

3.6.8.2.9.17.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.2.9.17.6 DECOMPOSITION

None.

3.6.8.2.9.18 VECTOR_MATRIX_MULTIPLY_UNRESTRICTED (CATALOG #P185-0)

This package contains a function which multiplies a 1 x m vector by an m x n matrix producing a 1 x n vector. If the length of the vector is not the same as the length of the first dimension of the matrix a DIMENSION_ERROR exception

is raised. None of the ranges need to be the same.

The function in this package can be made to handle sparse matrices and/or vectors by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

The following exceptions are raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the length of the vector is not the same as the length of the first dimension of the input matrix

3.6.8.2.9.18.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.18.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Input_Vector_Elements	floating point type	Type of elements in the input vector
Matrix_Elements	floating point type	Type of elements in the input matrix
Output_Vector_Elements	floating point type	Type of elements in the output vector
Col_Indices	discrete type	Used to dimension second dimension of input matrix
Row_Indices	discrete type	Used to dimension first dimension of input matrix
Input_Vector_Indices	discrete	Used to dimension input vector
Output_Vector_Indices	discrete	Used to dimension output vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Input_Vector_Elements * Matrix_Elements := Output_Vector_Elements
"+"	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements

3.6.8.2.9.18.3 LOCAL ENTITIES

None.

3.6.8.2.9.18.4 INTERRUPTS

None.

3.6.8.2.9.18.5 TIMING AND SEQUENCING

The following shows a sample usage of this part where neither the input matrix or vector are sparse:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   type My_Elements3_Squared is new My_Elements3;
   ...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   type My_Indices     is (i, j, k);
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;

   function "*" (Left  : My_Elements2,
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function "*" (Left  : My_Elements3,
                 Right : My_Elements3) return My_Elements3_Squared;
   ...
   function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
   function Sqrt (Input : My_Elements3_Squared) return My_Elements3;
   ...
   package V_Opns1 is new
           General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
              (Vector_Elements         => My_Elements1,
               Vector_Elements_Squared => My_Elements1_Squared,
               Indices                 => My_Indices);
```

```
   use V_Opns1;
   ...
   subtype Vectors1 is V_Opns1.Vectors(My_Indices);
   ...
   package M_Opns2 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements2,
               Row_Indices => My_Row_Indices);
   use M_Opns2;
   ...
   subtype Matrices2 is M_Opns2.Matrices(My_Row_Indices, My_Col_Indices);
   ...
   package V_Opns3 is new
           General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
              (Vector_Elements         => My_Elements3,
               Vector_Elements_Squared => My_Elements3_Squared,
               Indices                 => My_Indices);
   use V_Opns3;
   ...
   subtype Vectors3 is V_Opns3.Vectors(My_Indices);
   ...
   function "*" is new
            General_Vector_Matrix_Algebra.
               Vector_Matrix_Multiply_Unrestricted
                  (Matrix_Elements        => My_Elements2,
                   Input_Vector_Elements  => My_Elements1,
                   Output_Vector_Elements => My_Elements3,
                   Col_Indices            => My_Col_Indices,
                   Input_Vector_Indices   => My_Indices,
                   Output_Vector_Indices  => My_Indices,
                   Row_Indices            => My_Row_Indices,
                   Input_Vectors          => Vectors1,
                   Input_Matrices         => Matrices2,
                   Output_Vectors         => Vectors3);
   ...
   Vector1 : Vectors1;
   Matrix2 : Matrices2;
   Vector3 : Vectors3;
   ...
   begin
   ...
      Vector3 := Vector1 * Matrix2;
```

The following shows a sample usage of this part where the input matrix is sparse:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   type My_Elements3_Squared is new My_Elements3;
   ...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
```

```
type My_Indices     is (i, j, k);
...
function Sparse_Left_Multiply
             (Left  : My_Elements1;
              Right : My_Elements2) return My_Elements3;
...
function Sparse_Left_Add
             (Left  : My_Elements3;
              Right : My_Elements3) return My_Elements3;
...
function "*" (Left  : My_Elements2,
              Right : My_Elements2) return My_Elements2_Squared;
...
function "*" (Left  : My_Elements3,
              Right : My_Elements3) return My_Elements3_Squared;
...
function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
function Sqrt (Input : My_Elements3_Squared) return My_Elements3;
...
package V_Opns1 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements         => My_Elements1,
            Vector_Elements_Squared => My_Elements1_Squared,
            Indices                 => My_Indices);
use V_Opns1;
...
package Sparse_M_Opns2 is new
        General_Vector_Matrix_Algebra.
        Dynamically_Sparse_Matrix_Operations_Unconstrained
           (Col_Indices => My_Col_Indices,
            Elements    => My_Elements2,
            Row_Indices => My_Row_Indices);
use Sparse_M_Opns2;
...
subtype Sp_Matrices2 is Sparse_M_Opns2.
                        Matrices(My_Row_Indices, My_Col_Indices);
...
package V_Opns3 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements         => My_Elements3,
            Vector_Elements_Squared => My_Elements3_Squared,
            Indices                 => My_Indices);
use V_Opns3;
...
subtype Vectors3 is V_Opns3.Vectors(My_Indices);
...
package V_M_Mult is new
        General_Vector_Matrix_Algebra.
           Vector_Matrix_Multiply_Unrestricted
              (Matrix_Elements        => My_Elements2,
               Input_Vector_Elements  => My_Elements1,
               Output_Vector_Elements => My_Elements3,
               Col_Indices            => My_Col_Indices,
               Input_Vector_Indices   => My_Indices,
               Output_Vector_Indices  => My_Indices,
               Row_Indices            => My_Row_Indices,
               Input_Vectors          => Vectors1,
```

```
                  Input_Matrices         => Sp_Matrices2,
                  Output_Vectors         => Vectors3,
                  "*"                    => Sparse_Left_Multiply,
                  "+"                    => Sparse_Left_Add);
   use V_M_Mult;
   ...
   Vector1    : Vectors1;
   Sp_Matrix2 : Sp_Matrices2;
   Vector3    : Vectors3;
   ...
   function Sparse_Left_Multiply
               (Left  : My_Elements1;
                Right : My_Elements2) return My_Elements3 is
      Answer : My_Elements3;
   begin
      if Left = 0.0 then
         Answer := My_Elements3(Right);
      else
         Answer := My_Elements3(Left * My_Elements1(Right));
      end if;
      return Answer;
   end Sparse_Left_Multiply;
   ...
   function Sparse_Left_Add
               (Left  : My_Elements3;
                Right : My_Elements3) return My_Elements3 is
      Answer : My_Elements3;
   begin
      if Left = 0.0 then
         Answer := Right;
      else
         Answer := Left + Right;
      end if;
      return Answer;
   end Sparse_Left_Add;
   ...
   begin
   ...
      Vector3 := Vector1 * Sp_Matrix2;
```

3.6.8.2.9.18.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.18.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies a 1 x m vector by an m x n matrix, returning the resultant 1 x n vector c(j) := a(i) * b(i,j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P1052-0

3.6.8.2.9.18.8 PART DESIGN

None.

3.6.8.2.9.19 VECTOR_MATRIX_MULTIPLY_RESTRICTED (CATALOG #P186-0)

This package contains a function which multiplies a 1 x m vector by an m x n matrix producing a 1 x n vector.

The calculations performed are as follows:

```
c(j) := a(i) * b(i,j)
```

The function can be made to handle sparse matrices and/or vectors by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

3.6.8.2.9.19.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.19.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Indices1	discrete type	Used to dimension first dimension of input matrix and to dimension the output vector
Indices2	discrete type	Used to dimension second dimension of input matrix and to dimension the input vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Input_Vector_Elements * Matrix_Elements := Output_Vector_Elements
"+"	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector	Input_Vectors	In	M-element vector
Matrix	Input_Matrices	In	M x N input matrix

3.6.8.2.9.19.3 INTERRUPTS

None.

3.6.8.2.9.19.4 TIMING AND SEQUENCING

The following shows a sample usage of this part where neither the input matrix or vector are sparse:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   type My_Elements3_Squared is new My_Elements3;
   ...
   type My_Indices1 is (x, y, z);
   type My_Indices2 is (a, b, c);
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;

   function "*" (Left  : My_Elements2,
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function "*" (Left  : My_Elements3,
                 Right : My_Elements3) return My_Elements3_Squared;
   ...
   function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
   function Sqrt (Input : My_Elements3_Squared) return My_Elements3;
   ...
   package V_Opns1 is new
           General_Vector_Matrix_Algebra.Vector_Operations_Constrained
              (Vector_Elements         => My_Elements1,
               Vector_Elements_Squared => My_Elements1_Squared,
               Indices                 => My_Indices1);
   use V_Opns1;
   ...
   subtype Vectors1 is V_Opns1.Vectors;
   ...
   package M_Opns2 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_Indices2,
               Elements    => My_Elements2,
               Row_Indices => My_Indices1);
   use M_Opns2;
   ...
   subtype Matrices2 is M_Opns2.Matrices;
   ...
   package V_Opns3 is new
           General_Vector_Matrix_Algebra.Vector_Operations_Constrained
              (Vector_Elements         => My_Elements3,
               Vector_Elements_Squared => My_Elements3_Squared,
               Indices                 => My_Indices2);
   use V_Opns3;
   ...
   subtype Vectors3 is V_Opns3.Vectors;
   ...
   function "*" is new
            General_Vector_Matrix_Algebra.
```

```
             Vector_Matrix_Multiply_Constrained
                (Matrix_Elements          => My_Elements2,
                 Input_Vector_Elements    => My_Elements1,
                 Output_Vector_Elements   => My_Elements3,
                 Indices1                 => My_Indices1,
                 Indices2                 => My_Indices2,
                 Input_Vectors            => Vectors1,
                 Input_Matrices           => Matrices2,
                 Output_Vectors           => Vectors3);
   ...
   Vector1 : Vectors1;
   Matrix2 : Matrices2;
   Vector3 : Vectors3;
   ...
   begin
   ...
      Vector3 := Vector1 * Matrix2;
```

The following shows a sample usage of this part where the input matrix is sparse:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   type My_Elements3_Squared is new My_Elements3;
   ...
   type My_Indices1 is (x, y, z);
   type My_Indices2 is (a, b, c);
   ...
   function Sparse_Left_Multiply
             (Left  : My_Elements1;
              Right : My_Elements2) return My_Elements3;
   ...
   function Sparse_Left_Add
             (Left  : My_Elements3;
              Right : My_Elements3) return My_Elements3;
   ...
   function "*" (Left  : My_Elements2,
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function "*" (Left  : My_Elements3,
                 Right : My_Elements3) return My_Elements3_Squared;
   ...
   function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
   function Sqrt (Input : My_Elements3_Squared) return My_Elements3;
   ...
   package V_Opns1 is new
           General_Vector_Matrix_Algebra.Vector_Operations_Constrained
              (Vector_Elements         => My_Elements1,
               Vector_Elements_Squared => My_Elements1_Squared,
               Indices                 => My_Indices1);
   use V_Opns1;
   ...
   subtype Vectors1 is V_Opns1.Vectors;
```

```
...
package Sparse_M_Opns2 is new
        General_Vector_Matrix_Algebra.
        Dynamically_Sparse_Matrix_Operations_Constrained
           (Col_Indices => My_Indices2,
            Elements    => My_Elements2,
            Row_Indices => My_Indices1);
use Sparse_M_Opns2;
...
subtype Sp_Matrices2 is Sparse_M_Opns2.Matrices;
...
package V_Opns3 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Constrained
           (Vector_Elements         => My_Elements3,
            Vector_Elements_Squared => My_Elements3_Squared,
            Indices                 => My_Indices2);
use V_Opns3;
...
subtype Vectors3 is V_Opns3.Vectors(My_Indices);
...
package V_M_Mult is new
        General_Vector_Matrix_Algebra.
           Vector_Matrix_Multiply_Restricted
              (Matrix_Elements        => My_Elements2,
               Input_Vector_Elements  => My_Elements1,
               Output_Vector_Elements => My_Elements3,
               Indices1               => My_Indices1,
               Indices2               => My_Indices2,
               Input_Matrices         => Sp_Matrices2,
               Input_Vectors          => Vectors1,
               Output_Vectors         => Vectors3,
               "*"                    => Sparse_Left_Multiply,
               "+"                    => Sparse_Left_Add);
use V_M_Mult;
...
Vector1    : Vectors1;
Sp_Matrix2 : Sp_Matrices2;
Vector3    : Vectors3;
...
function Sparse_Left_Multiply
            (Left  : My_Elements1;
             Right : My_Elements2) return My_Elements3 is
   Answer : My_Elements3;
begin
   if Left = 0.0 then
      Answer := My_Elements3(Right);
   else
      Answer := My_Elements3(Left * My_Elements1(Right));
   end if;
   return Answer;
end Sparse_Left_Multiply;
...
function Sparse_Left_Add
            (Left  : My_Elements3;
             Right : My_Elements3) return My_Elements3 is
   Answer : My_Elements3;
begin
```

```
      if Left = 0.0 then
         Answer := Right;
      else
         Answer := Left + Right;
      end if;
      return Answer;
   end Sparse_Left_Add;
   ...
   begin
   ...
      Vector3 := Vector2 * Sp_Matrix1;
```

3.6.8.2.9.19.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.2.9.19.6 DECOMPOSITION

None.

3.6.8.2.9.20 VECTOR_VECTOR_TRANSPOSE_MULTIPLY_UNRESTRICTED (CATALOG #P187-0)

This package contains a function which multiplies an m x 1 vector by the transpose of an n x 1 vector, returning the resultant m x n matrix.

The following exceptions are raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the length of the left input vector is not the same as the length of the first dimension of the output matrix or if the length of the right input vector is not the same as the second dimension of the output matrix

3.6.8.2.9.20.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.20.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Vector_Elements	floating point type	Data type of elements in left input vector
Right_Vector_Elements	floating point type	Data type of elements in right input vector
Matrix_Elements	floating point type	Data type of elements in output matrix
Left_Vector_Indices	discrete	Used to dimension left input vector
Right_Vector_Indices	discrete	Used to dimension right input vector
Col_Indices	discrete type	Used to dimension second dimension of output matrix
Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector
Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the multiplication operation Left_Vector_Elements * Right_Vector_Elements := Matrix_Elements

3.6.8.2.9.20.3 LOCAL ENTITIES

None.

3.6.8.2.9.20.4 INTERRUPTS

None.

3.6.8.2.9.20.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements1_Squared is new My_Elements1;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   ...
   type My_Col_Indices is (a, b, c);
```

```
type My_Row_Indices is (x, y, z);
type My_Indices     is (i, j, k);
...
function "*" (Left  : My_Elements1,
              Right : My_Elements2) return My_Elements3;
...
function "*" (Left  : My_Elements1,
              Right : My_Elements1) return My_Elements1_Squared;
...
function "*" (Left  : My_Elements2,
              Right : My_Elements2) return My_Elements2_Squared;
...
function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
...
package V_Opns1 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements         => My_Elements1,
            Vector_Elements_Squared => My_Elements1_Squared,
            Indices                 => My_Indices);
use V_Opns1;
...
subtype Vectors1 is V_Opns1.Vectors(My_Indices);
...
package V_Opns2 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements         => My_Elements2,
            Vector_Elements_Squared => My_Elements2_Squared,
            Indices                 => My_Indices);
use V_Opns2;
...
subtype Vectors2 is V_Opns2.Vectors(My_Indices);
...
package M_Opns3 is new
        General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
           (Col_Indices => My_Col_Indices,
            Elements    => My_Elements3,
            Row_Indices => My_Row_Indices);
use M_Opns3;
...
subtype Matrices3 is M_Opns3.Matrices(My_Row_Indices, My_Col_Indices);
...
package V_V_T_Multiply is new
           General_Vector_Matrix_Algebra.
              Vector_Vector_Transpose_Multiply_Unrestricted
                 (Left_Vector_Elements   => My_Elements1,
                  Matrix_Elements        => My_Elements3,
                  Right_Vector_Elements  => My_Elements2,
                  Col_Indices            => My_Col_Indices,
                  Left_Vector_Indices    => My_Indices,
                  Right_Vector_Indices   => My_Indices,
                  Row_Indices            => My_Row_Indices,
                  Left_Vectors           => Vectors1,
                  Input_Matrices         => Matrices3,
                  Right_Vectors          => Vectors2);
use V_V_T_Multiply;
...
```

```
   Vector1 : Vectors1;
   Vector2 : Vectors2;
   Matrix3 : Matrices3;
   ...
   begin
   ...
      Matrix3 := Vector1 * Vector2;
```

3.6.8.2.9.20.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.20.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies an m x 1 vector by the transpose of an n x 1 vector, returning the resultant m x n matrix c(i,j) := a(i) * b(j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P532-0

3.6.8.2.9.20.8 PART DESIGN

None.

3.6.8.2.9.21 VECTOR_VECTOR_TRANSPOSE_MULTIPLY_RESTRICTED (CATALOG #P188-0)

This function multiplies an m x 1 vector by the transpose of an n x 1 vector, returning the resultant m x n matrix.

3.6.8.2.9.21.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.21.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Vector_Elements	floating point type	Data type of elements in left input vector
Right_Vector_Elements	floating point type	Data type of elements in right input vector
Matrix_Elements	floating point type	Data type of elements in output matrix
Indices1	discrete type	Used to dimension left input vector and first dimension of output matrix
Indices2	discrete type	Used to dimension right input vector and second dimension of output matrix
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector
Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the multiplication operation Left_Vector_Elements * Right_Vector_Elements := Matrix_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Vectors	In	Vector to be used as the multiplicand
Right	Right_Vectors	In	Vector whose transpose is to be used as the multiplier

3.6.8.2.9.21.3 INTERRUPTS

None.

3.6.8.2.9.21.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements1_Squared is new My_Elements1;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   ...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements1) return My_Elements1_Squared;
   ...
   function "*" (Left  : My_Elements2,
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
   function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
   ...
   package V_Opns1 is new
           General_Vector_Matrix_Algebra.Vector_Operations_Constrained
              (Vector_Elements         => My_Elements1,
               Vector_Elements_Squared => My_Elements1_Squared,
               Indices                 => My_Row_Indices);
   use V_Opns1;
   ...
   subtype Vectors1 is V_Opns1.Vectors(My_Indices);
   ...
   package V_Opns2 is new
           General_Vector_Matrix_Algebra.Vector_Operations_Constrained
              (Vector_Elements         => My_Elements2,
               Vector_Elements_Squared => My_Elements2_Squared,
               Indices                 => My_Indices);
   use V_Opns2;
   ...
   subtype Vectors2 is V_Opns2.Vectors(My_Indices);
   ...
   package M_Opns3 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements3,
               Row_Indices => My_Row_Indices);
   use M_Opns3;
   ...
   subtype Matrices3 is M_Opns3.Matrices(My_Row_Indices, My_Col_Indices);
   ...
   function V_V_T_Multiply is new
               General_Vector_Matrix_Algebra.
                  Vector_Vector_Transpose_Multiply_Restricted
```

```
                    (Left_Vector_Elements   => My_Elements1,
                     Matrix_Elements        => My_Elements3,
                     Right_Vector_Elements  => My_Elements2,
                     Indices1               => My_Row_Indices,
                     Indices2               => My_Col_Indices,
                     Left_Vectors           => Vectors1,
                     Matrices               => Matrices3,
                     Right_Vectors          => Vectors2);
   use V_V_T_Multiply;
   ...
   Vector1 : Vectors1;
   Vector2 : Vectors2;
   Matrix3 : Matrices3;
   ...
   begin
   ...
      Matrix3 := Vector1 * Vector2;
```

3.6.8.2.9.21.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.2.9.21.6 DECOMPOSITION

None.

3.6.8.2.9.22 MATRIX_MATRIX_MULTIPLY_UNRESTRICTED (CATALOG #P189-0)

This package contains a function which multiplies an m x n matrix by an n x p matrix, returning an m x p matrix. The inner dimensions of the input matrices must be equal. This part can be made to handle sparse matrices by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

The following exceptions are raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the inner dimensions of the input matrices are not the same or if the lengths of the first dimension of the left input matrix and the second dimension of the right input matrix are not the same as the corresponding dimensions of the output matrix

3.6.8.2.9.22.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.

3.6.8.2.9.22.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Data type of elements in left input matrix
Right_Elements	floating point type	Data type of elements in right input matrix
Output_Elements	floating point type	Data type of elements in output matrix
Left_Col_Indices	discrete type	Used to dimension second dimension of left input matrix
Left_Row_Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_Indices	discrete type	Used to dimension second dimension of output matrix
Output_Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part. To tailor this function to handle sparse matrices, the formal subroutines should be set up to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Left_Elements * Right_Elements := Output_Elements
"+"	function	Function defining the operation Output_Elements + Output_Elements := Output_Elements

3.6.8.2.9.22.3 LOCAL ENTITIES

None.

541

3.6.8.2.9.22.4 INTERRUPTS

None.

3.6.8.2.9.22.5 TIMING AND SEQUENCING

The following shows a sample usage of this part assuming both input matrices are dense matrices:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1 is new FLOAT;
   type My_Elements2 is new FLOAT;
   type My_Elements3 is new FLOAT;
   ...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;
   ...
   package M_Opns1 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements1,
               Row_Indices => My_Row_Indices);
   use M_Opns1;
   ...
   subtype Matrices1 is M_Opns1.Matrices(My_Row_Indices, My_Col_Indices);
   ...
   package M_Opns2 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements2,
               Row_Indices => My_Row_Indices);
   use M_Opns2;
   ...
   subtype Matrices2 is M_Opns2.Matrices(My_Row_Indices, My_Col_Indices);
   ...
   package M_Opns3 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements3,
               Row_Indices => My_Row_Indices);
   use M_Opns3;
   ...
   subtype Matrices3 is M_Opns3.Matrices(My_Row_Indices, My_Col_Indices);
   ...
   package M_M_Mult is new
            General_Vector_Matrix_Algebra.
               Matrix_Matrix_Multiply_Unrestricted
                  (Left_Elements       => My_Elements1,
                   Output_Elements     => My_Elements3,
                   Right_Elements      => My_Elements2,
                   Left_Col_Indices    => My_Col_Indices,
                   Left_Row_Indices    => My_Row_Indices,
```

```
                    Output_Col_Indices => My_Col_Indices,
                    Output_Row_Indices => My_Row_Indices,
                    Right_Col_Indices  => My_Col_Indices,
                    Right_Row_Indices  => My_Row_Indices,
                    Left_Matrices      => Matrices1,
                    Output_Matrices    => Matrices3,
                    Right_Matrices     => Matrices2);
   use M_M_Mult;
   ...
   Matrix1 : Matrices1;
   Matrix2 : Matrices2;
   Matrix3 : Matrices3;
   ...
   begin
      ...
      Matrix3 := Matrix1 * Matrix2;
```

The following shows how this part could to tailored to handle the multiplication of sparse matrices. In this example, it is assumed that the left input matrix is the sparse matrix. This part could also be tailored to handle a sparse matrix for the right parameter or sparse matrices for both parameters by modifying the multiplication and addition operators.

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1 is new FLOAT;
   type My_Elements2 is new FLOAT;
   type My_Elements3 is new FLOAT;
   ...
   type My_Col_Indices is (a, b, c);
   type My_Row_Indices is (x, y, z);
   ...
   function Sparse_Left_Multiply
            (Left  : My_Elements1,
             Right : My_Elements2) return My_Elements3;
   ...
   function Sparse_Left_Add
            (Left  : My_Elements3,
             Right : My_Elements3) return My_Elements3;
   ...
   package Dyn_Sparse_M_Opns1 is new
           General_Vector_Matrix_Algebra.
              Dynamically_Sparse_Matrix_Operations_Unconstrained
                 (Col_Indices => My_Col_Indices,
                  Elements    => My_Elements1,
                  Row_Indices => My_Row_Indices);
   use Dyn_Sparse_M_Opns1;
   ...
   subtype Sp_Matrices1 is Dyn_Sparse_M_Opns1.
                                Matrices(My_Row_Indices, My_Col_Indices);
   ...
   package M_Opns2 is new
           General_Vector_Matrix_Algebra.
              Matrix_Operations_Unconstrained
                 (Col_Indices => My_Col_Indices,
                  Elements    => My_Elements2,
```

```
                Row_Indices => My_Row_Indices);
use M_Opns2;
...
subtype Matrices2 is M_Opns2.Matrices(My_Row_Indices, My_Col_Indices);
...
package Dyn_Sparse_M_Opns3 is new
        General_Vector_Matrix_Algebra.
           Dynamically_Sparse_Matrix_Operations_Unconstrained
              (Col_Indices => My_Col_Indices,
               Elements    => My_Elements3,
               Row_Indices => My_Row_Indices);
use Dyn_Sparse_M_Opns3;
...
subtype Sp_Matrices3 is Dyn_Sparse_M_Opns3.
                           Matrices(My_Row_Indices, My_Col_Indices);
...
function "*" is new
         General_Vector_Matrix_Algebra.
            Matrix_Matrix_Multiply_Unrestricted
               (Left_Elements        => My_Elements1,
                Output_Elements      => My_Elements3,
                Right_Elements       => My_Elements2,
                Left_Col_Indices     => My_Col_Indices,
                Left_Row_Indices     => My_Row_Indices,
                Output_Col_Indices   => My_Col_Indices,
                Output_Row_Indices   => My_Row_Indices,
                Right_Col_Indices    => My_Col_Indices,
                Right_Row_Indices    => My_Row_Indices,
                Left_Matrices        => Sp_Matrices1,
                Output_Matrices      => Sp_Matrices3,
                Right_Matrices       => Matrices2,
                "*"                  => Sparse_Left_Multiply,
                "+"                  => Sparse_Left_Add);
...
Sp_Matrix1 : Sp_Matrices1;
Matrix2    : Matrices2;
Sp_Matrix3 : Sp_Matrices3;
...
function Sparse_Left_Multiply
            (Left  : My_Elements1,
             Right : My_Elements2) return My_Elements3 is
   Answer : My_Elements3;
begin
   if Left = 0.0 then
      Answer := My_Elements3(Right);
   else
      Answer := My_Elements3(Left * My_Elements1(Answer));
   end if;
   return Answer;
end Sparse_Left_Multiply;
...
function Sparse_Left_Add
            (Left  : My_Elements3,
             Right : My_Elements3) return My_Elements3
   Answer : My_Elements3;
begin
   if Left = 0.0 then
```

```
         Answer := Right;
      else
         Answer := Left + Right;
      end if;
      return Answer;
   end Sparse_Left_Add;
   ...
   begin
      ...
      Sp_Matrix3 := Sp_Matrix1 * Matrix2;
```

3.6.8.2.9.22.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.22.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies an m x n matrix by an n x p matrix, returning the result m x p matrix c(i,j) := a(i,k) * b(k,j)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P533-0

3.6.8.2.9.22.8 PART DESIGN

None.

3.6.8.2.9.23 MATRIX_MATRIX_MULTIPLY_RESTRICTED (CATALOG #P190-0)

This function multiplies an m x n matrix by an n x p matrix, returning an m x p matrix. This part can be made to handle sparse matrices by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

3.6.8.2.9.23.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.

3.6.8.2.9.23.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Data type of elements in left input matrix
Right_Elements	floating point type	Data type of elements in right input matrix
Output_Elements	floating point type	Data type of elements in output matrix
M_Indices	discrete type	Used to dimension first dimension of left input matrix and output matrix
N_Indices	discrete type	Used to dimension second dimension of left input matrix and first dimension of right input matrix
P_Indices	discrete type	Used to dimension second dimension of right input matrix and output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part. To tailor this function to handle sparse matrices, the formal subroutines should be set up to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation Left_Elements * Right_Elements := Output_Elements
"+"	function	Function defining the operation Output_Elements + Output_Elements := Output_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Matrices	In	M X N matrix to be used as the multiplicand
Right	Right_Matrices	In	N X P matrix to be used as the multiplier

3.6.8.2.9.23.3 INTERRUPTS

None.

3.6.8.2.9.23.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

The following shows a sample usage of this part assuming both input matrices are dense matrices:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1 is new FLOAT;
   type My_Elements2 is new FLOAT;
   type My_Elements3 is new FLOAT;
   ...
   type My_M_Indices is new INTEGER 1..3;
   type My_N_Indices is new INTEGER 1..4;
   type My_P_Indices is new INTEGER 1..3;
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;
   ...
   package M_Opns1 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_N_Indices,
               Elements    => My_Elements1,
               Row_Indices => My_M_Indices);
   use M_Opns1;
   ...
   subtype Matrices1 is M_Opns1.Matrices;
   ...
   package M_Opns2 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_P_Indices,
               Elements    => My_Elements2,
               Row_Indices => My_N_Indices);
   use M_Opns2;
   ...
   subtype Matrices2 is M_Opns2.Matrices;
   ...
   package M_Opns3 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_P_Indices,
               Elements    => My_Elements3,
```

```
            Row_Indices => My_M_Indices);
   use M_Opns3;
   ...
   subtype Matrices3 is M_Opns3.Matrices;
   ...
   package M_M_Mult is new
            General_Vector_Matrix_Algebra.
               Matrix_Matrix_Multiply_Restricted
                  (Left_Elements       => My_Elements1,
                   Output_Elements     => My_Elements3,
                   Right_Elements      => My_Elements2,
                   M_Indices           => My_M_Indices,
                   N_Indices           => My_N_Indices,
                   P_Indices           => My_P_Indices,
                   Left_Matrices       => Matrices1,
                   Output_Matrices     => Matrices3,
                   Right_Matrices      => Matrices2);
   use M_M_Mult;
   ...
   Matrix1 : Matrices1;
   Matrix2 : Matrices2;
   Matrix3 : Matrices3;
   ...
   begin
      ...
      Matrix3 := Matrix1 * Matrix2;
```

The following shows how this part could to tailored to handle the multiplication of sparse matrices. In this example, it is assumed that the left input matrix is the sparse matrix. This part could also be tailored to handle a sparse matrix for the right parameter or sparse matrices for both parameters by modifying the multiplication and addition operators.

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1 is new FLOAT;
   type My_Elements2 is new FLOAT;
   type My_Elements3 is new FLOAT;
   ...
   type My_M_Indices is new INTEGER 1..3;
   type My_N_Indices is new INTEGER 1..4;
   type My_P_Indices is new INTEGER 1..3;
   ...
   function Sparse_Left_Multiply
               (Left  : My_Elements1,
                Right : My_Elements2) return My_Elements3;
   ...
   function Sparse_Left_Add
               (Left  : My_Elements3,
                Right : My_Elements3) return My_Elements3;
   ...
   package Dyn_Sparse_M_Opns1 is new
            General_Vector_Matrix_Algebra.
               Dynamically_Sparse_Matrix_Operations_Constrained
                  (Col_Indices => My_N_Indices,
                   Elements    => My_Elements1,
```

```
                 Row_Indices => My_M_Indices);
use Dyn_Sparse_M_Opns1;
...
subtype Sp_Matrices1 is Dyn_Sparse_M_Opns1.Matrices;
...
package M_Opns2 is new
        General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
           (Col_Indices => My_P_Indices,
            Elements    => My_Elements2,
            Row_Indices => My_N_Indices);
use M_Opns2;
...
subtype Matrices2 is M_Opns2.Matrices;
...
package Dyn_Sparse_M_Opns3 is new
        General_Vector_Matrix_Algebra.
           Dynamically_Sparse_Matrix_Operations_Constrained
              (Col_Indices => My_P_Indices,
               Elements    => My_Elements3,
               Row_Indices => My_M_Indices);
use Dyn_Sparse_M_Opns3;
...
subtype Sp_Matrices3 is Dyn_Sparse_M_Opns3.Matrices;
...
function "*" is new
         General_Vector_Matrix_Algebra.
            Matrix_Matrix_Multiply_Restricted
               (Left_Elements      => My_Elements1,
                Output_Elements    => My_Elements3,
                Right_Elements     => My_Elements2,
                M_Indices          => My_M_Indices,
                N_Indices          => My_N_Indices,
                P_Indices          => My_P_Indices,
                Left_Matrices      => Sp_Matrices1,
                Output_Matrices    => Sp_Matrices3,
                Right_Matrices     => Matrices2,
                "*"                => Sparse_Left_Multiply,
                "+"                => Sparse_Left_Add);
...
Sp_Matrix1 : Sp_Matrices1;
Matrix2    : Matrices2;
Sp_Matrix3 : Sp_Matrices3;
...
function Sparse_Left_Multiply
           (Left  : My_Elements1,
            Right : My_Elements2) return My_Elements3 is
   Answer : My_Elements3;
begin
   if Left = 0.0 then
      Answer := My_Elements3(Right);
   else
      Answer := My_Elements3(Left * My_Elements1(Answer));
   end if;
   return Answer;
end Sparse_Left_Multiply;
...
function Sparse_Left_Add
```

```
              (Left  : My_Elements3,
               Right : My_Elements3) return My_Elements3
      Answer : My_Elements3;
   begin
      if Left = 0.0 then
         Answer := Right;
      else
         Answer := Left + Right;
      end if;
      return Answer;
   end Sparse_Left_Add;
   ...
   begin
      ...
      Sp_Matrix3 := Sp_Matrix1 * Matrix2;
```

3.6.8.2.9.23.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.2.9.23.6 DECOMPOSITION

None.

3.6.8.2.9.24 MATRIX_MATRIX_TRANSPOSE_MULTIPLY_UNRESTRICTED (CATALOG #P191-0)

This package contains a function which multiplies an m x n matrix by the transpose of a p x n matrix returning the resultant m x p matrix.

The following exceptions are raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the second dimensions of the two input matrices are not of the same length, if the lengths of the first dimensions of the left and output matrices are not the same, or if the lengths of the second dimensions of the right and output matrices are not the same

3.6.8.2.9.24.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.24.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input matrix
Right_Elements	floating point type	Type of elements in right input matrix
Output_Elements	floating point type	Type of elements in output matrix
Left_Col_ Indices	discrete type	Used to dimension second dimension of left input matrix
Left_Row_ Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_ Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_ Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_ Indices	discrete type	Used to dimension second dimension of output matrix
Output_Row_ Indices	discrete type	Used to dimension first dimension of output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator used to define the operation: Left_Elements * Right_Elements := Output_Elements

3.6.8.2.9.24.3 LOCAL ENTITIES

None.

3.6.8.2.9.24.4 INTERRUPTS

None.

3.6.8.2.9.24.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
```

```
type My_Elements1 is new FLOAT;
type My_Elements2 is new FLOAT;
type My_Elements3 is new FLOAT;
...
type My_Col_Indices is (a, b, c);
type My_Row_Indices is (x, y, z);
...
function "*" (Left  : My_Elements1,
              Right : My_Elements2) return My_Elements3;
...
package M_Opns1 is new
        General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
           (Col_Indices => My_Col_Indices,
            Elements    => My_Elements1,
            Row_Indices => My_Row_Indices);
use M_Opns1;
...
subtype Matrices1 is M_Opns1.Matrices(My_Row_Indices, My_Col_Indices);
...
package M_Opns2 is new
        General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
           (Col_Indices => My_Col_Indices,
            Elements    => My_Elements2,
            Row_Indices => My_Row_Indices);
use M_Opns2;
...
subtype Matrices2 is M_Opns2.Matrices(My_Row_Indices, My_Col_Indices);
...
package M_Opns3 is new
        General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
           (Col_Indices => My_Col_Indices,
            Elements    => My_Elements3,
            Row_Indices => My_Row_Indices);
use M_Opns3;
...
subtype Matrices3 is M_Opns3.Matrices(My_Row_Indices, My_Col_Indices);
...
package M_M_T_Mult is new
         General_Vector_Matrix_Algebra.
            Matrix_Matrix_Transpose_Multiply_Unrestricted
               (Left_Elements      => My_Elements1,
                Output_Elements    => My_Elements3,
                Right_Elements     => My_Elements2,
                Left_Col_Indices   => My_Col_Indices,
                Left_Row_Indices   => My_Row_Indices,
                Output_Col_Indices => My_Col_Indices,
                Output_Row_Indices => My_Row_Indices,
                Right_Col_Indices  => My_Col_Indices,
                Right_Row_Indices  => My_Row_Indices,
                Left_Matrices      => Matrices1,
                Output_Matrices    => Matrices3,
                Right_Matrices     => Matrices2);
use M_M_T_Mult;
...
Matrix1 : Matrices1;
Matrix2 : Matrices2;
Matrix3 : Matrices3;
```

```
...
begin
   ...
   Matrix3 := Matrix1 * Matrix2;
```

3.6.8.2.9.24.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.24.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"*"	function	Multiplies an m x n matrix by the transpose of an p x n matrix, returning the resultant m x p matrix a(i,j) := b(i,k) * c(j,k)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"*"	P534-0

3.6.8.2.9.24.8 PART DESIGN

None.

3.6.8.2.9.25 MATRIX_MATRIX_TRANSPOSE_MULTIPLY_RESTRICTED (CATALOG #P192-0)

This function multiplies an m x n matrix by the transpose of a p x n matrix returning the resultant m x p matrix.

3.6.8.2.9.25.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.25.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input matrix
Right_Elements	floating point type	Type of elements in right input matrix
Output_Elements	floating point type	Type of elements in output matrix
M_Indices	discrete type	Used to dimension first dimension of left input matrix and output matrix
N_Indices	discrete type	Used to dimension second dimension of left and right input matrix
P_Indices	discrete type	Used to dimension first dimension of right input matrix and second dimension of output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator used to define the operation: Left_Elements * Right_Elements := Output_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Matrices	In	M X N matrix to be used as the multiplicand
Right	Right_Matrices	In	P X N matrix whose transpose is to be used as the multiplier

3.6.8.2.9.25.3 INTERRUPTS

None.

3.6.8.2.9.25.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1 is new FLOAT;
   type My_Elements2 is new FLOAT;
   type My_Elements3 is new FLOAT;
   ...
   type My_M_Indices is (a, b, c);
   type My_N_Indices is (x, y, z);
   type My_P_Indices is (x, y, z);
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;
   ...
   package M_Opns1 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_N_Indices,
               Elements    => My_Elements1,
               Row_Indices => My_M_Indices);
   use M_Opns1;
   ...
   subtype Matrices1 is M_Opns1.Matrices;
   ...
   package M_Opns2 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_N_Indices,
               Elements    => My_Elements2,
               Row_Indices => My_P_Indices);
   use M_Opns2;
   ...
   subtype Matrices2 is M_Opns2.Matrices;
   ...
   package M_Opns3 is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_P_Indices,
               Elements    => My_Elements3,
               Row_Indices => My_M_Indices);
   use M_Opns3;
   ...
   subtype Matrices3 is M_Opns3.Matrices;
   ...
   package M_M_T_Mult is new
            General_Vector_Matrix_Algebra.
               Matrix_Matrix_Transpose_Multiply_Restricted
                  (Left_Elements      => My_Elements1,
                   Output_Elements    => My_Elements3,
                   Right_Elements     => My_Elements2,
                   M_Indices          => My_M_Indices,
                   N_Indices          => My_N_Indices,
                   P_Indices          => My_P_Indices,
                   Left_Matrices      => Matrices1,
                   Output_Matrices    => Matrices3,
                   Right_Matrices     => Matrices2);
   use M_M_T_Mult;
```

```
...
Matrix1 : Matrices1;
Matrix2 : Matrices2;
Matrix3 : Matrices3;
...
begin
   ...
   Matrix3 := Matrix1 * Matrix2;
```

3.6.8.2.9.25.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.2.9.25.6 DECOMPOSITION

None.

3.6.8.2.9.26 DOT_PRODUCT_OPERATIONS_UNRESTRICTED (CATALOG #P193-0)

This package contains a function which performs a dot product operation on two m-element vectors.

The following exceptions are raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the two input vectors is not the same

3.6.8.2.9.26.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R063.

3.6.8.2.9.26.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input vector
Right_Elements	floating point type	Type of elements in right input vector
Result_Elements	floating point type	Data type of result of dot product
Left_Indices	discrete	Used to dimension Left_Vectors
Right_Indices	discrete	Used to dimension Right_Vectors
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Left_Elements * Right_Elements := Result_Elements

3.6.8.2.9.26.3 LOCAL ENTITIES

None.

3.6.8.2.9.26.4 INTERRUPTS

None.

3.6.8.2.9.26.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements1_Squared is new My_Elements1;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   ...
   type My_Indices     is (i, j, k);
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements1) return My_Elements1_Squared;
```

```
...
function "*" (Left  : My_Elements2,
              Right : My_Elements2) return My_Elements2_Squared;
...
function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
...
package V_Opns1 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements          => My_Elements1,
            Vector_Elements_Squared  => My_Elements1_Squared,
            Indices                  => My_Indices);
use V_Opns1;
...
subtype Vectors1 is V_Opns1.Vectors(My_Indices);
...
package V_Opns2 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
           (Vector_Elements          => My_Elements2,
            Vector_Elements_Squared  => My_Elements2_Squared,
            Indices                  => My_Indices);
use V_Opns2;
...
subtype Vectors2 is V_Opns2.Vectors(My_Indices);
...
package D_P_Opns is new
         General_Vector_Matrix_Algebra.
            Dot_Product_Operations_Unrestricted
            (Left_Elements           => My_Elements1,
             Right_Elements          => My_Elements2,
             Result_Elements         => My_Elements3,
             Left_Indices            => My_Indices,
             Right_Indices           => My_Indices,
             Left_Vectors            => Vectors1,
             Right_Vectors           => Vectors2);
...
Vector1 : Vectors1;
Vector2 : Vectors2;
Temp    : My_Elements3;
...
begin
...
   Temp := D_P_Opns.Dot_Product(Vector1, Vector2);
```

3.6.8.2.9.26.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.26.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Dot_Product	function	Performs a dot product operation on two m-element vectors c := a(i) + b(i)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
Dot_Product	P535-0

3.6.8.2.9.26.8 PART DESIGN

None.

3.6.8.2.9.27 DOT_PRODUCT_OPERATIONS_RESTRICTED (CATALOG #P194-0)

This function performs a dot product operation on two m-element vectors.

3.6.8.2.9.27.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R063.

3.6.8.2.9.27.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input vector
Right_Elements	floating point type	Type of elements in right input vector
Result_Elements	floating point type	Data type of result of dot product
Indices	discrete type	Used to dimension input vectors
Left_Vectors	array	Data type of left input vector
Right_Vectors	array	Data type of right input vector

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Left_Elements * Right_Elements := Result_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Vectors	In	First vector to be used in dot product operation
Right	Right_Vectors	In	Second vector to be used in dot product operation

3.6.8.2.9.27.3 INTERRUPTS

None.

3.6.8.2.9.27.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Elements1         is new FLOAT;
   type My_Elements1_Squared is new My_Elements1;
   type My_Elements2         is new FLOAT;
   type My_Elements2_Squared is new My_Elements2;
   type My_Elements3         is new FLOAT;
   ...
   type My_Indices     is (i, j, k);
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements2) return My_Elements3;
   ...
   function "*" (Left  : My_Elements1,
                 Right : My_Elements1) return My_Elements1_Squared;
   ...
   function "*" (Left  : My_Elements2,
                 Right : My_Elements2) return My_Elements2_Squared;
   ...
   function Sqrt (Input : My_Elements1_Squared) return My_Elements1;
   function Sqrt (Input : My_Elements2_Squared) return My_Elements2;
   ...
```

```
package V_Opns1 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Constrained
           (Vector_Elements         => My_Elements1,
            Vector_Elements_Squared => My_Elements1_Squared,
            Indices                 => My_Indices);
use V_Opns1;
...
subtype Vectors1 is V_Opns1.Vectors;
...
package V_Opns2 is new
        General_Vector_Matrix_Algebra.Vector_Operations_Constrained
           (Vector_Elements         => My_Elements2,
            Vector_Elements_Squared => My_Elements2_Squared,
            Indices                 => My_Indices);
use V_Opns2;
...
subtype Vectors2 is V_Opns2.Vectors;
...
function Dot_Product D_P_Opns is new
         General_Vector_Matrix_Algebra.
            Dot_Product_Operations_Restricted
            (Left_Elements          => My_Elements1,
             Right_Elements         => My_Elements2,
             Result_Elements        => My_Elements3,
             Indices                => My_Indices,
             Left_Vectors           => Vectors1,
             Right_Vectors          => Vectors2);
...
Vector1 : Vectors1;
Vector2 : Vectors2;
Temp    : My_Elements3;
...
begin
...
   Temp := Dot_Product(Vector1, Vector2);
```

3.6.8.2.9.27.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.2.9.27.6 DECOMPOSITION

None.

3.6.8.2.9.28 DIAGONAL_FULL_MATRIX_ADD_UNRESTRICTED (CATALOG #P195-0)

This package contains a function which adds a diagonal matrix to a full matrix, returning the resultant full matrix.

The diagonal matrix is represented as a one-dimensional matrix. While this part is designed to expect a diagonal matrix as defined by the Diagonal_-Matrix_Operations package. This part can be instantiated using the Diagonal_-Matrices type exported by an instantiated version of the Diagonal_Matrix_-Operations package or with a similarly-defined matrix declared by the user.

941

The following exceptions are raised by this part:

Name	When/Why Raised
Dimension_Error	Raised if the lengths of the matrix indices are not equal to each other and other the length of the diagonal matrix

3.6.8.2.9.28.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.6.8.2.9.28.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Type of elements in input and output arrays
Diagonal_Range	integer type	Used to dimension Diagonal_Matrices
Full_Input_Col_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Input_Row_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Output_Col_Indices	discrete	Used to dimension Full_Output_ matrices
Full_Output_Row_Indices	discrete	Used to dimension Full_Output_ matrices
Diagonal_Matrices	array	Data type of diagonal input matrix
Full_Input_Matrices	array	Data type of full input matrix
Full_Output_Matrices	array	Data type of full output matrix

3.6.8.2.9.28.3 LOCAL ENTITIES

None.

3.6.8.2.9.28.4 INTERRUPTS

None.

3.6.8.2.9.28.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Col_Indices1 is (a, b, c);
   type My_Row_Indices1 is (x, y, z);
   type My_Col_Indices2 is (i, j, k);
   type My_Row_Indices2 is (l, m, n);
   ...
   type My_Elements         is new FLOAT;
   type My_Elements_Squared is new FLOAT;
   ...
   function "*" (Left  : My_Elements;
                 Right : My_Elements) return My_Elements_Squared;
   ...
   function Sqrt (Input : My_Elements_Squared) return My_Elements;
   ...
   package V_Opns_XYZ is new
           General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
              (Vector_Elements         => My_Elements,
               Vector_Elements_Squared => My_Elements_Squared,
               Indices                 => My_Row_Indices1);
   use V_Opns_XYZ;
   ...
   subtype Vectors_XYZ is V_Opns_XYZ.Vectors(My_Row_Indices1);
   ...
   package V_Opns_ABC is new
           General_Vector_Matrix_Algebra.Vector_Operations_Unconstrained
              (Vector_Elements         => My_Elements,
               Vector_Elements_Squared => My_Elements_Squared,
               Indices                 => My_Col_Indices1);
   use V_Opns_ABC;
   ...
   subtype Vectors_ABC is V_Opns_ABC.Vectors(My_Col_Indices);
   ...
   package Diagonal_M_Opns is new
           General_Vector_Matrix_Algebra.
              Diagonal_Matrix_Operations
                 (Elements     => My_Elements,
                  Col_Indices  => My_Col_Indices1,
                  Row_Indices  => My_Row_Indices1,
                  Col_Slices   => Vectors_XYZ,
                  Row_Slices   => Vectors_ABC);
   use Diagonal_M_Opns;
   ...
   package M_Opns is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Unconstrained
              (Col_Indices => My_Col_Indices2,
               Elements    => My_Elements,
               Row_Indices => My_Row_Indices2);
   use M_Opns;
   ...
   subtype Matrices is M_Opns.Matrices(My_Row_Indices2, My_Col_Indices2);
   ...
   package D_F_M_Add is new
```

```
            General_Vector_Matrix_Algebra.Diagonal_Full_Matrix_Add_Unrestricted
               (Elements                  => My_Elements,
                Diagonal_Range            => D_Opns.Diagonal_Range,
                Full_Input_Col_Indices    => My_Col_Indices2,
                Full_Input_Row_Indices    => My_Row_Indices2,
                Full_Output_Col_Indices   => My_Col_Indices2,
                Full_Output_Row_Indices   => My_Row_Indices2,
                Diagonal_Matrices         => D_Opns.Diagonal_Matrices,
                Full_Input_Matrices       => Matrices,
                Full_Output_Matrices      => Matrices);
   use D_F_M_Add;
   ...
   Diag_Matrix : Diagonal_M_Opns.Diagonal_Matrices;
   Full_Matrix : Matrices;
   ...
   begin
      ...
      Full_Matrix := Diag_Matrix + Full_Matrix;
```

3.6.8.2.9.28.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.28.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
"+"	function	Adds an m x m full matrix to an m-element diagonal matrix

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
"+"	P536-0

3.6.8.2.9.28.8 PART DESIGN

None.

3.6.8.2.9.29 DIAGONAL_FULL_MATRIX_ADD_RESTRICTED (CATALOG #P196-0).

This function adds a diagonal matrix to a full matrix, returning the resultant full matrix.

The diagonal matrix is represented as a one-dimensional matrix. This part can be instantiated using the Diagonal_Matrices type exported by an instantiated version of the Diagonal_Matrix_Operations package or with a similarly-defined matrix declared by the user.

3.6.8.2.9.29.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.6.8.2.9.29.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Type of elements in input and output arrays
Diagonal_Range	integer type	Used to dimension Diagonal_Matrices
Indices	discrete type	Used to dimension input and output matrices
Diagonal_Matrices	array	Data type of diagonal input matrix
Full_Matrices	array	Data type of full input and output matrices

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
D_Matrix	Diagonal_Matrices	In	Diagonal matrix to be used in the addition operation
F_Matrix	Full_Matrices	In	Full matrix to beused in the addition operation

3.6.8.2.9.29.3 INTERRUPTS

None.

3.6.8.2.9.29.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
```

```
...
   type My_Indices is (a, b, c);
   ...
   type My_Elements         is new FLOAT;
   type My_Elements_Squared is new FLOAT;
   ...
   function "*" (Left  : My_Elements;
                 Right : My_Elements) return My_Elements_Squared;
   ...
   function Sqrt (Input : My_Elements_Squared) return My_Elements;
   ...
   package V_Opns is new
           General_Vector_Matrix_Algebra.Vector_Operations_Constrained
              (Vector_Elements         => My_Elements,
               Vector_Elements_Squared => My_Elements_Squared,
               Indices                 => My_Indices);
   use V_Opns;
   ...
   subtype My_Vectors is V_Opns.Vectors;
   ...
   package Diagonal_M_Opns is new
           General_Vector_Matrix_Algebra.
              Diagonal_Matrix_Operations
                 (Elements       => My_Elements,
                  Col_Indices    => My_Indices,
                  Row_Indices    => My_Indices,
                  Col_Slices     => My_Vectors,
                  Row_Slices     => My_Vectors);
   use Diagonal_M_Opns;
   ...
   package M_Opns is new
           General_Vector_Matrix_Algebra.Matrix_Operations_Constrained
              (Col_Indices => My_Indices,
               Elements    => My_Elements,
               Row_Indices => My_Indices);
   use M_Opns;
   ...
   subtype Matrices is M_Opns.Matrices;
   ...
   functin "+" is new
              General_Vector_Matrix_Algebra.
                 Diagonal_Full_Matrix_Add_Restricted
                    (Elements          => My_Elements,
                     Diagonal_Range    => D_Opns.Diagonal_Range,
                     Indices           => My_Indices,
                     Diagonal_Matrices => D_Opns.Diagonal_Matrices,
                     Full_Matrices     => Matrices);
   ...
   Diag_Matrix : Diagonal_M_Opns.Diagonal_Matrices;
   Full_Matrix : Matrices;
   ...
   begin
      ...
      Full_Matrix := Diag_Matrix + Full_Matrix;
```

3.6.8.2.9.29.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.2.9.29.6 DECOMPOSITION

None.

3.6.8.2.9.30 ABA_TRANS_DYNAM_SPARSE_MATRIX_SQ_MATRIX (CATALOG #P1055-0)

This package contains a function which does an ABA transpose multiply on a dynamically sparse matrix (m x n) and a square (n x n) matrix, yielding a square matrix. The first multiply (A*B) is constrained and the second (AB * transpose A) is restricted.

3.6.8.2.9.30.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.30.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
A_Elements	floating point type	Type of element in the dymamically sparse input matrix
B_Elements	floating point type	Type of element in the square input matrix
C_Elements	floating point type	Type of element in the output vector
M_Indices	discrete type	Used to dimension the 1st dimension of the sparse input matrix
N_Indices	discrete type	Used to dimension the 2nd dimension of the sparse matrix and both dimensions of the square matrix
A_Matrices	array	Data type of the dynamically sparse input matrix
B_Matrices	array	Data type of the square input matrix
C_Matrices	array	Data type of the output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function defining the operation A_Elements * B_Elements := C_Elements
"*"	function	Function defining the operation C_Elements * A_Elements := C_Elements

3.6.8.2.9.30.3 LOCAL ENTITIES

None.

3.6.8.2.9.30.4 INTERRUPTS

None.

3.6.8.2.9.30.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_ElementsA is new FLOAT;
   type My_ElementsB is new FLOAT;
   type My_ElementsC is new FLOAT;
   ...
   type My_M_Indices is new INTEGER 1..3;
   type My_N_Indices is new INTEGER 1..4;
   ...
   type My_A_Matrices is array( My_M_Indices, My_N_Indices ) of My_ElementsA;
   type My_B_Matrices is array( My_N_Indices, My_N_Indices ) of My_ElementsB;
   type My_C_Matrices is array( My_M_Indices, My_M_Indices ) of My_ElementsC;
   ...
   function AB_Multiply
              (Left  : My_ElementsA,
               Right : My_ElementsB) return My_ElementsC;
   ...
   function CA_Multiply
              (Left  : My_ElementsC,
               Right : My_ElementsC) return My_ElementsC;
   ...
   package My_ABA_Transpose is new
           General_Vector_Matrix_Algebra.
              ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix
                ( A_Elements  => My_ElementsA,
                  B_Elements  => My_ElementsB,
                  C_Elements  => My_ElementsC,
                  M_Indices   => My_M_Indices,
                  N_Indices   => My_N_Indices,
                  A_Matrices  => My_A_Matrices,
```

```
                    B_Matrices  => My_B_Matrices,
                    C_Matrices  => My_C_Matrices,
                    "*"         => AB_Multiply,
                    "*"         => CA_Multiply );

   use My_ABA_Transpose;
   ...
   Matrix1 : A_Matrices;
   Matrix2 : B_Matrices;
   Matrix3 : C_Matrices;
   ...
   begin
      ...
      Matrix3 := My_ABA_Transpose( Left  => Matrix1,
                                   Right => Matrix2 );
```

3.6.8.2.9.30.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.30.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
ABA_Transpose	function	Multiplies an m x n dynamically sparse matrix * an n x n square matrix and multiplies that product by the transpose of the m x n matrix

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
ABA_Transpose	P1056-0

3.6.8.2.9.30.8 PART DESIGN

None.

3.6.8.2.9.31 ABA_TRANS_VECTOR_SQ_MATRIX (CATALOG #P1057-0)

This package contains a function which does an ABA transpose multiply on a vector (1 x m) and a square (m x m) matrix, yielding a scalar value.

3.6.8.2.9.31.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.31.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Vector_ Elements	floating point type	Type of element in the input vector.
Matrix_ Elements	floating point type	Type of element in the square input matrix.
Scalars	floating point type	Type of element in the output scalar
Indices	discrete type	Used to dimension the input vector and both dimensions of the input matrix
Vectors	array	Data type of the input vector
Matrices	array	Data type of the square input matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function defining the operation Vector_Elements * Matrix_Elements := Vector_Elements
"*"	function	Function defining the operation Vector_Elements * Vector_Elements := Scalars

3.6.8.2.9.31.3 LOCAL ENTITIES

None.

3.6.8.2.9.31.4 INTERRUPTS

None.

THIS REPORT HAS BEEN DELIMITED AND CLEARED FOR PUBLIC RELEASE UNDER DOD DIRECTIVE 5200.20 AND NO RESTRICTIONS ARE IMPOSED UPON ITS USE AND DISCLOSURE.

DISTRIBUTION STATEMENT A

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

3.6.8.2.9.31.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Vector_Elements is new FLOAT;
   type My_Matrix_Elements is new FLOAT;
   type My_Scalars         is new FLOAT;
   ...
   type My_Indices is new INTEGER 1..3;
   ...
   type My_Matrices is array( My_Indices, My_Indices ) of My_Matrix_Elements;
   type My_Vectors is array( My_Indices ) of My_Vector_Elements;
   ...
   function AB_Multiply
             (Left  : My_Vector_Elements,
              Right : My_Matrices ) return My_Vector_Elements;
   ...
   function CA_Multiply
             (Left  : My_Vector_Elements,
              Right : My_Vector_Elements ) return My_Scalars;
   ...
   package My_ABA_Transpose is new
           General_Vector_Matrix_Algebra.
              ABA_Trans_Vector_Sq_Matrix
                 ( Vector_Elements  => My_Vector_Elements,
                   Matrix_Elements  => My_Matrix_Elements,
                   Scalars          => My_Scalars,
                   Indices          => My_Indices,
                   Vectors          => My_Vectors,
                   Matrices         => My_Matrices,
                   "*"              => AB_Multiply,
                   "*"              => CA_Multiply );

   use My_ABA_Transpose;
   ...
   My_Vector : My_Vectors;
   My_Matrix : My_Matrices;
   My_Scalar : My_Scalars;
   ...
   begin
      ...
      My_Scalar := My_ABA_Transpose( Left  => My_Vector,
                                     Right => My_Matrix );
```

3.6.8.2.9.31.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.31.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
ABA_Transpose	function	Multiplies an 1 x m vector * a square matrix (m x m) and multiplies the resultant vector (1 x m) times the transpose of the original vector (m x 1) yielding a scalar (1 x 1)

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
ABA_Transpose	058-0

3.6.8.2.9.31.8 PART DESIGN

None.

3.6.8.2.9.32 ABA_TRANS_VECTOR_SCALAR (CATALOG #P1059-0)

This package contains a function which does an ABA transpose multiply on a vector (m x 1) and a scalar value, yielding a square (m x m) matrix.

3.6.8.2.9.32.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.32.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Vector_ Elements	floating point type	Type of element in the input vector.
Matrix_ Elements	floating point type	Type of element in the square output matrix.
Scalars	floating point type	Type of element in the output scalar
Indices	discrete type	Used to dimension the input vector and the square output matrix
Vectors	array	Data type of the input vector
Matrices	array	Data type of the square output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Function defining the operation Vector_Elements * Scalars := Vector_Elements
"*"	function	Function defining the operation Vector_Elements * Vector_Elements := Matrix_Elements

3.6.8.2.9.32.3 LOCAL ENTITIES

None.

3.6.8.2.9.32.4 INTERRUPTS

None.

3.6.8.2.9.32.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Vector_Elements is new FLOAT;
   type My_Matrix_Elements is new FLOAT;
   type My_Scalars         is new FLOAT;
   ...
   type My_Indices is new INTEGER 1..3;
   ...
   type My_Matrices is array( My_Indices, My_Indices ) of My_Matrix_Elements;
   type My_Vectors is array( My_Indices ) of My_Vector_Elements;
   ...
   function AB_Multiply
              (Left  : My_Vector_Elements,
               Right : My_Scalars ) return My_Vector_Elements;
   ...
   function CA_Multiply
              (Left  : My_Vector_Elements,
               Right : My_Vector_Elements ) return My_Matrix_Elements;
   ...
   package My_ABA_Transpose is new
           General_Vector_Matrix_Algebra.
              ABA_Trans_Vector_Scalar
                 ( Vector_Elements  => My_Vector_Elements,
                   Matrix_Elements  => My_Matrix_Elements,
                   Scalars          => My_Scalars,
                   Indices          => My_Indices,
                   Vectors          => My_Vectors,
                   Matrices         => My_Matrices,
```

```
                  "*"                => AB_Multiply,
                  "*"                => CA_Multiply );

   use My_ABA_Transpose;
   ...
   My_Vector : My_Vectors;
   My_Matrix : My_Matrices;
   My_Scalar : My_Scalars;
   ...
   begin
      ...
      My_Matrix := My_ABA_Transpose( Left  => My_Vector,
                                     Right => My_Scalar );
```

3.6.8.2.9.32.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.32.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
ABA_Transpose	function	Multiplies an m x 1 vector * a scalar and multiplies the resultant m x 1 vector times the transpose (1 x m) of the original vector, yielding an m x m square matrix.

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
ABA_Transpose	P1060-0

3.6.8.2.9.32.8 PART DESIGN

None.

3.6.8.2.9.33 COLUMN_MATRIX_OPERATIONS (CATALOG #P1061-0)

This package defines a column matrix which contains a column vector which is set on one of the columns of the matrix and a diagonal, which can only have the values of 1 or 0 on the diagonal. It provides operations on that type. See the decomposition section for a list of the operations provided.

3.6.8.2.9.33.1 REQUIREMENTS ALLOCATION

N/A.

3.6.8.2.9.33.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Vector_ Elements	floating point type	Type of element in the column matrix's column vector
Indices	discrete type	Used to dimension the column matrix and the vector in the column matrix
Vectors	array	Data type of the vector in the column matrix

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data objects:

The following chart describes the data objects exported by this part:

Name	Type	Description
Column_ Matrices	record	Record in which the following information describing the column matrix is kept:
Col_ Vector	Vectors	The column vector
Diagonal	BOOLEAN	A BOOLEAN value which tells whether the diagonal is an identity matrix (contains 1's) or is 0's
Active_ Column	Indices	An index which identifies the column number where the column vector is to be set

3.6.8.2.9.33.3 LOCAL ENTITIES

None.

3.6.8.2.9.33.4 INTERRUPTS

None.

3.6.8.2.9.33.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Vector_Matrix_Algebra;
...
   type My_Vector_Elements is new FLOAT;
   ...
   type My_Indices is new INTEGER 1..3;
   ...
   type My_Vectors is array( My_Indices ) of My_Vector_Elements;
   ...
   package My_Col_Matrix_Opns is new
           General_Vector_Matrix_Algebra.
              Column_Matrix_Operations
                 ( Vector_Elements   => My_Vector_Elements,
                   Indices           => My_Indices,
                   Vectors           => My_Vectors );

   use My_Col_Matrix_Opns;
   ...
   Vector     : My_Vectors;
   Col_Matrix : My_Col_Matrix_Opns.Column_Matrices;
   Index      : My_Indices;
   ...
   begin
      ...
      --  assign values to vector and index
      ...
      Col_Matrix := Set_Column_With_Zeroes_On_Diagonal
         ( Column        => Vector;
           Active_Column => Index );
```

3.6.8.2.9.33.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.2.9.33.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Set_Column_With_Zeroes_On_Diagonal	function	Sets (assigns) the values of the input vector to the column vector, designates the active_column in the matrix, and sets the diagonal BOOLEAN to false
Subtract_From_Identity	function	Subtracts the input column matrix from the identity matrix (1's on the diagonal)
ABA_Transpose	generic function	Does an A * B * A Transpose operation on a Column matrix and a square matrix
ABA_Symm_Transpose	generic function	Does an (A * B) * A Transpose operation on a Column matrix and a symmetric full storage matrix

The following table lists the allocation of catalog numbers to this part:

Name	Catalog #
Set_Column_Diagonal	P1062-0
Subtract_From_Identity	P1063-0
ABA_Transpose	P1064-0
ABA_Sym_Trans	P1065-0

3.6.8.2.9.33.8 PART DESIGN

None.

(This page left intentionally blank.)

```
package General_Vector_Matrix_Algebra is

   Dimension_Error : exception;

pragma PAGE;
   generic
      type Vector_Elements         is digits <>;
      type Vector_Elements_Squared is digits <>;
      type Indices                 is (<>);
      with function "*"  (Left  : Vector_Elements;
                          Right : Vector_Elements)
                         return Vector_Elements_Squared is <>;
      with function Sqrt (Input : Vector_Elements_Squared)
                         return Vector_Elements is <>;
   package Vector_Operations_Unconstrained is

      type Vectors is array (Indices range <>) of Vector_Elements;

      function "+" (Left  : Vectors;
                    Right : Vectors) return Vectors;

      function "-" (Left  : Vectors;
                    Right : Vectors) return Vectors;

      function Dot_Product (Left  : Vectors;
                            Right : Vectors) return Vector_Elements_Squared;

      function Vector_Length (Input : Vectors) return Vector_Elements;

   end Vector_Operations_Unconstrained;

pragma PAGE;
   generic
      type Vector_Elements         is digits <>;
      type Vector_Elements_Squared is digits <>;
      type Indices                 is (<>);
      with function "*"  (Left  : Vector_Elements;
                          Right : Vector_Elements)
                         return Vector_Elements_Squared is <>;
      with function Sqrt (Input : Vector_Elements_Squared)
                         return Vector_Elements is <>;
   package Vector_Operations_Constrained is

      type Vectors is array (Indices) of Vector_Elements;

      function "+" (Left  : Vectors;
                    Right : Vectors) return Vectors;

      function "-" (Left  : Vectors;
                    Right : Vectors) return Vectors;

      function Dot_Product (Left  : Vectors;
                            Right : Vectors) return Vector_Elements_Squared;

      function Vector_Length (Input : Vectors) return Vector_Elements;

   end Vector_Operations_Constrained;
```

```
pragma PAGE;
   generic
      type Elements    is digits <>;
      type Col_Indices is (<>);
      type Row_Indices is (<>);
   package Matrix_Operations_Unconstrained is

      type Matrices is array (Row_Indices range <>,
                              Col_Indices range <>) of Elements;

      function "+" (Left  : Matrices;
                    Right : Matrices) return Matrices;

      function "-" (Left  : Matrices;
                    Right : Matrices) return Matrices;

      function "+" (Matrix : Matrices;
                    Addend : Elements) return Matrices;

      function "-" (Matrix     : Matrices;
                    Subtrahend : Elements) return Matrices;

      procedure Set_To_Identity_Matrix (Matrix : out Matrices);

      procedure Set_To_Zero_Matrix (Matrix : out Matrices);

      function "*" (Left  : Matrices;
                    Right : Matrices) return Matrices;

   end Matrix_Operations_Unconstrained;

pragma PAGE;
   generic
      type Elements    is digits <>;
      type Col_Indices is (<>);
      type Row_Indices is (<>);
   package Matrix_Operations_Constrained is

      type Matrices is array (Row_Indices,
                              Col_Indices) of Elements;

      function "+" (Left  : Matrices;
                    Right : Matrices) return Matrices;

      function "-" (Left  : Matrices;
                    Right : Matrices) return Matrices;

      function "+" (Matrix : Matrices;
                    Addend : Elements) return Matrices;

      function "-" (Matrix     : Matrices;
                    Subtrahend : Elements) return Matrices;

      procedure Set_To_Identity_Matrix (Matrix : out Matrices);

      procedure Set_To_Zero_Matrix (Matrix : out Matrices);
```

```
   end Matrix_Operations_Constrained;

pragma PAGE;
   generic
      type Elements    is digits <>;
      type Col_Indices is (<>);
      type Row_Indices is (<>);
   package Dynamically_Sparse_Matrix_Operations_Unconstrained is

      type Matrices is array (Row_Indices range <>,
                              Col_Indices range <>) of Elements;

      procedure Set_To_Identity_Matrix (Matrix : out Matrices);

      procedure Set_To_Zero_Matrix     (Matrix : out Matrices);

      function Add_To_Identity (Input : Matrices) return Matrices;

      function Subtract_From_Identity (Input : Matrices) return Matrices;

      function "+" (Left  : Matrices;
                    Right : Matrices) return Matrices;

      function "-" (Left  : Matrices;
                    Right : Matrices) return Matrices;

   end Dynamically_Sparse_Matrix_Operations_Unconstrained;

pragma PAGE;
   generic
      type Elements    is digits <>;
      type Col_Indices is (<>);
      type Row_Indices is (<>);
   package Dynamically_Sparse_Matrix_Operations_Constrained is

      type Matrices is array (Row_Indices, Col_Indices) of Elements;

      procedure Set_To_Identity_Matrix (Matrix : out Matrices);

      procedure Set_To_Zero_Matrix     (Matrix : out Matrices);

      function Add_To_Identity (Input : Matrices) return Matrices;

      function Subtract_From_Identity (Input : Matrices) return Matrices;

      function "+" (Left  : Matrices;
                    Right : Matrices) return Matrices;

      function "-" (Left  : Matrices;
                    Right : Matrices) return Matrices;

   end Dynamically_Sparse_Matrix_Operations_Constrained;

pragma PAGE;
   generic
      type Elements    is digits <>;
```

```
      type Col_Indices is (<>);
      type Row_Indices is (<>);
      type Col_Slices  is array (Row_Indices) of Elements;
      type Row_Slices  is array (Col_Indices) of Elements;
   package Symmetric_Half_Storage_Matrix_Operations is

      Entry_Count : constant POSITIVE
                  := Row_Slices'LENGTH * (Row_Slices'LENGTH + 1) / 2;

      type Matrices is array (1..Entry_Count) of Elements;

      procedure Initialize (Row_Slice : in     Row_Slices;
                            Row       : in     Row_Indices;
                            Matrix    :    out Matrices);

      function Identity_Matrix return Matrices;

      function Zero_Matrix     return Matrices;

      procedure Change_Element (New_Value : in     Elements;
                                Row       : in     Row_Indices;
                                COL       : in     Col_Indices;
                                Matrix    :    out Matrices);

      function Retrieve_Element (Matrix : Matrices;
                                 Row    : Row_Indices;
                                 COL    : Col_Indices) return Elements;

      function Row_Slice (Matrix : Matrices;
                          Row    : Row_Indices) return Row_Slices;

      function Column_Slice (Matrix : Matrices;
                             COL    : Col_Indices) return Col_Slices;

      function Add_To_Identity        (Input : Matrices) return Matrices;

      function Subtract_From_Identity (Input : Matrices) return Matrices;

      function "+" (Left  : Matrices;
                    Right : Matrices) return Matrices;

      function "-" (Left  : Matrices;
                    Right : Matrices) return Matrices;

   end Symmetric_Half_Storage_Matrix_Operations;

pragma PAGE;
   generic
      type Elements    is digits <>;
      type Col_Indices is (<>);
      type Row_Indices is (<>);
   package Symmetric_Full_Storage_Matrix_Operations_Unconstrained is

      Invalid_Index : exception;

      type Matrices is array (Row_Indices range <>,
                              Col_Indices range <>) of Elements;
```

```
      procedure Change_Element (New_Value : in     Elements;
                                Row       : in     Row_Indices;
                                COL       : in     Col_Indices;
                                Matrix    : in out Matrices);

      procedure Set_To_Identity_Matrix (Matrix : out Matrices);

      procedure Set_To_Zero_Matrix (Matrix : out Matrices);

      function Add_To_Identity (Input : Matrices) return Matrices;

      function Subtract_From_Identity (Input : Matrices) return Matrices;

      function "+" (Left  : Matrices;
                    Right : Matrices) return Matrices;

      function "-" (Left  : Matrices;
                    Right : Matrices) return Matrices;

   end Symmetric_Full_Storage_Matrix_Operations_Unconstrained;

pragma PAGE;
   generic
      type Elements    is digits <>;
      type Col_Indices is (<>);
      type Row_Indices is (<>);
   package Symmetric_Full_Storage_Matrix_Operations_Constrained is

      type Matrices is array (Row_Indices, Col_Indices) of Elements;

      procedure Change_Element (New_Value : in     Elements;
                                Row       : in     Row_Indices;
                                COL       : in     Col_Indices;
                                Matrix    : in out Matrices);

      procedure Set_To_Identity_Matrix (Matrix : out Matrices);

      procedure Set_To_Zero_Matrix (Matrix : out Matrices);

      function Add_To_Identity (Input : Matrices) return Matrices;

      function Subtract_From_Identity (Input : Matrices) return Matrices;

      function "+" (Left  : Matrices;
                    Right : Matrices) return Matrices;

      function "-" (Left  : Matrices;
                    Right : Matrices) return Matrices;

   end Symmetric_Full_Storage_Matrix_Operations_Constrained;

pragma PAGE;
   generic
      type Elements       is digits <>;
      type Col_Indices    is (<>);
      type Row_Indices    is (<>);
```

```
      type Col_Slices    is array (Row_Indices)           of Elements;
      type Row_Slices    is array (Col_Indices)           of Elements;
   package Diagonal_Matrix_Operations is

      Invalid_Index : exception;

      Entry_Count : constant POSITIVE := Row_Slices'LENGTH;

      subtype Diagonal_Range is POSITIVE range 1..Entry_Count;

      type Diagonal_Matrices is array (Diagonal_Range) of Elements;

      function Identity_Matrix return Diagonal_Matrices;

      function Zero_Matrix     return Diagonal_Matrices;

      procedure Change_Element (New_Value : in     Elements;
                                Row       : in     Row_Indices;
                                COL       : in     Col_Indices;
                                Matrix    :    out Diagonal_Matrices);

      function Retrieve_Element (Matrix  : Diagonal_Matrices;
                                 Row     : Row_Indices;
                                 COL     : Col_Indices) return Elements;

      function Row_Slice (Matrix : Diagonal_Matrices;
                          Row    : Row_Indices) return Row_Slices;

      function Column_Slice (Matrix : Diagonal_Matrices;
                             COL    : Col_Indices) return Col_Slices;

      function Add_To_Identity (Input : Diagonal_Matrices)
                                return Diagonal_Matrices;

      function Subtract_From_Identity (Input : Diagonal_Matrices)
                                       return Diagonal_Matrices;

      function "+" (Left  : Diagonal_Matrices;
                    Right : Diagonal_Matrices) return Diagonal_Matrices;

      function "-" (Left  : Diagonal_Matrices;
                    Right : Diagonal_Matrices) return Diagonal_Matrices;

   end Diagonal_Matrix_Operations;

pragma PAGE;
   generic
      type Elements1 is digits <>;
      type Elements2 is digits <>;
      type Scalars   is digits <>;
      type Indices1  is (<>);
      type Indices2  is (<>);
      type Vectors1  is array(Indices1 range <>) of Elements1;
      type Vectors2  is array(Indices2 range <>) of Elements2;
      with function "*" (Left  : Elements2;
                         Right : Scalars) return Elements1 is <>;
      with function "/" (Left  : Elements1;
```

```
                         Right : Scalars) return Elements2 is <>;
   package Vector_Scalar_Operations_Unconstrained is

      function "*" (Vector     : Vectors2;
                    Multiplier : Scalars) return Vectors1;

      function "/" (Vector : Vectors1;
                    Divisor : Scalars) return Vectors2;

   end Vector_Scalar_Operations_Unconstrained;

pragma PAGE;
   generic
      type Elements1 is digits <>;
      type Elements2 is digits <>;
      type Scalars   is digits <>;
      type Indices   is (<>);
      type Vectors1  is array(Indices) of Elements1;
      type Vectors2  is array(Indices) of Elements2;
      with function "*" (Left  : Elements2;
                         Right : Scalars) return Elements1 is <>;
      with function "/" (Left  : Elements1;
                         Right : Scalars) return Elements2 is <>;
   package Vector_Scalar_Operations_Constrained is

      function "*" (Vector     : Vectors2;
                    Multiplier : Scalars) return Vectors1;

      function "/" (Vector : Vectors1;
                    Divisor : Scalars) return Vectors2;

   end Vector_Scalar_Operations_Constrained;

pragma PAGE;
   generic
      type Elements1    is digits <>;
      type Elements2    is digits <>;
      type Scalars      is digits <>;
      type Col_Indices1 is (<>);
      type Row_Indices1 is (<>);
      type Col_Indices2 is (<>);
      type Row_Indices2 is (<>);
      type Matrices1    is array (Row_Indices1 range <>,
                                  Col_Indices1 range <>) of Elements1;
      type Matrices2    is array (Row_Indices2 range <>,
                                  Col_Indices2 range <>) of Elements2;
      with function "*" (Left  : Elements1;
                         Right : Scalars) return Elements2 is <>;
      with function "/" (Left  : Elements2;
                         Right : Scalars) return Elements1 is <>;
   package Matrix_Scalar_Operations_Unconstrained is

      function "*" (Matrix     : Matrices1;
                    Multiplier : Scalars) return Matrices2;

      function "/" (Matrix  : Matrices2;
                    Divisor : Scalars) return Matrices1;
```

```
   end Matrix_Scalar_Operations_Unconstrained;

pragma PAGE;
   generic
      type Elements1    is digits <>;
      type Elements2    is digits <>;
      type Scalars      is digits <>;
      type Col_Indices  is (<>);
      type Row_Indices  is (<>);
      type Matrices1    is array (Row_Indices, Col_Indices) of Elements1;
      type Matrices2    is array (Row_Indices, Col_Indices) of Elements2;
      with function "*" (Left  : Elements1;
                         Right : Scalars) return Elements2 is <>;
      with function "/" (Left  : Elements2;
                         Right : Scalars) return Elements1 is <>;
   package Matrix_Scalar_Operations_Constrained is

      function "*" (Matrix     : Matrices1;
                    Multiplier : Scalars) return Matrices2;

      function "/" (Matrix  : Matrices2;
                    Divisor : Scalars) return Matrices1;

   end Matrix_Scalar_Operations_Constrained;

pragma PAGE;
   generic
      type Elements1          is digits <>;
      type Elements2          is digits <>;
      type Scalars            is digits <>;
      type Diagonal_Range1    is range <>;
      type Diagonal_Range2    is range <>;
      type Diagonal_Matrices1 is array(Diagonal_Range1) of Elements1;
      type Diagonal_Matrices2 is array(Diagonal_Range2) of Elements2;
      with function "*" (Left  : Elements1;
                         Right : Scalars) return Elements2 is <>;
      with function "/" (Left  : Elements2;
                         Right : Scalars) return Elements1 is <>;
   package Diagonal_Matrix_Scalar_Operations is

      function "*" (Matrix     : Diagonal_Matrices1;
                    Multiplier : Scalars) return Diagonal_Matrices2;

      function "/" (Matrix  : Diagonal_Matrices2;
                    Divisor : Scalars) return Diagonal_Matrices1;

   end Diagonal_Matrix_Scalar_Operations;

pragma PAGE;
   generic
      type Matrix_Elements        is digits <>;
      type Input_Vector_Elements  is digits <>;
      type Output_Vector_Elements is digits <>;
      type Col_Indices            is (<>);
      type Row_Indices            is (<>);
      type Input_Vector_Indices   is (<>);
```

```
      type Output_Vector_Indices  is (<>);
      type Input_Matrices is array (Row_Indices,
                                    Col_Indices) of Matrix_Elements;
      type Input_Vectors  is array (Input_Vector_Indices) of Input_Vector_Elements;
      type Output_Vectors is array (Output_Vector_Indices) of Output_Vector_Elements;
      with function "*" (Left  : Matrix_Elements;
                         Right : Input_Vector_Elements)
                        return Output_Vector_Elements is <>;
      with function "+" (Left  : Output_Vector_Elements;
                         Right : Output_Vector_Elements)
                        return Output_Vector_Elements is <>;
   package Matrix_Vector_Multiply_Unrestricted is

      function "*" (Matrix : Input_Matrices;
                    Vector : Input_Vectors) return Output_Vectors;

   end Matrix_Vector_Multiply_Unrestricted;

pragma PAGE;
   generic
      type Matrix_Elements        is digits <>;
      type Input_Vector_Elements  is digits <>;
      type Output_Vector_Elements is digits <>;
      type Indices1               is (<>);
      type Indices2               is (<>);
      type Input_Matrices is array (Indices1, Indices2) of Matrix_Elements;
      type Input_Vectors  is array (Indices2) of Input_Vector_Elements;
      type Output_Vectors is array (Indices1) of Output_Vector_Elements;
      with function "*" (Left  : Matrix_Elements;
                         Right : Input_Vector_Elements)
                        return Output_Vector_Elements is <>;
      with function "+" (Left  : Output_Vector_Elements;
                         Right : Output_Vector_Elements)
                        return Output_Vector_Elements is <>;
   function Matrix_Vector_Multiply_Restricted
             (Matrix : Input_Matrices;
              Vector : Input_Vectors) return Output_Vectors;

pragma PAGE;
   generic
      type Input_Vector_Elements  is digits <>;
      type Matrix_Elements        is digits <>;
      type Output_Vector_Elements is digits <>;
      type Input_Vector_Indices   is (<>);
      type Col_Indices            is (<>);
      type Row_Indices            is (<>);
      type Output_Vector_Indices  is (<>);
      type Input_Vectors  is array (Input_Vector_Indices) of Input_Vector_Elements;
      type Input_Matrices is array (Row_Indices,
                                    Col_Indices) of Matrix_Elements;
      type Output_Vectors is array (Output_Vector_Indices) of Output_Vector_Elements;
      with function "*" (Left  : Input_Vector_Elements;
                         Right : Matrix_Elements)
                        return Output_Vector_Elements is <>;
      with function "+" (Left  : Output_Vector_Elements;
                         Right : Output_Vector_Elements)
                        return Output_Vector_Elements is <>;
```

```
   package Vector_Matrix_Multiply_Unrestricted is

      function "*" (Vector : Input_Vectors;
                    Matrix : Input_Matrices) return Output_Vectors;

   end Vector_Matrix_Multiply_Unrestricted;

pragma PAGE;
   generic
      type Input_Vector_Elements  is digits <>;
      type Matrix_Elements        is digits <>;
      type Output_Vector_Elements is digits <>;
      type Indices1               is (<>);
      type Indices2               is (<>);
      type Input_Vectors  is array (Indices1) of Input_Vector_Elements;
      type Input_Matrices is array (Indices1, Indices2) of Matrix_Elements;
      type Output_Vectors is array (Indices2) of Output_Vector_Elements;
      with function "*" (Left  : Input_Vector_Elements;
                         Right : Matrix_Elements)
                        return Output_Vector_Elements is <>;
      with function "+" (Left  : Output_Vector_Elements;
                         Right : Output_Vector_Elements)
                        return Output_Vector_Elements is <>;
   function Vector_Matrix_Multiply_Restricted
              (Vector : Input_Vectors;
               Matrix : Input_Matrices) return Output_Vectors;

pragma PAGE;
   generic
      type Left_Vector_Elements  is digits <>;
      type Right_Vector_Elements is digits <>;
      type Matrix_Elements       is digits <>;
      type Left_Vector_Indices   is (<>);
      type Right_Vector_Indices  is (<>);
      type Col_Indices           is (<>);
      type Row_Indices           is (<>);
      type Left_Vectors  is array (Left_Vector_Indices)
                                of Left_Vector_Elements;
      type Right_Vectors is array (Right_Vector_Indices)
                                of Right_Vector_Elements;
      type Matrices      is array (Row_Indices,
                                   Col_Indices) of Matrix_Elements;
      with function "*" (Left  : Left_Vector_Elements;
                         Right : Right_Vector_Elements)
                        return Matrix_Elements is <>;
   package Vector_Vector_Transpose_Multiply_Unrestricted is

      function "*" (Left  : Left_Vectors;
                    Right : Right_Vectors) return Matrices;

   end Vector_Vector_Transpose_Multiply_Unrestricted;

pragma PAGE;
   generic
      type Left_Vector_Elements  is digits <>;
      type Right_Vector_Elements is digits <>;
      type Matrix_Elements       is digits <>;
```

```
      type Indices1            is (<>);
      type Indices2            is (<>);
      type Left_Vectors  is array (Indices1) of Left_Vector_Elements;
      type Right_Vectors is array (Indices2) of Right_Vector_Elements;
      type Matrices      is array (Indices1, Indices2) of Matrix_Elements;
      with function "*" (Left  : Left_Vector_Elements;
                         Right : Right_Vector_Elements)
                        return Matrix_Elements is <>;
   function Vector_Vector_Transpose_Multiply_Restricted
              (Left  : Left_Vectors ;
               Right : Right_Vectors) return Matrices;

pragma PAGE;
   generic
      type Left_Elements      is digits <>;
      type Right_Elements     is digits <>;
      type Output_Elements    is digits <>;
      type Left_Col_Indices   is (<>);
      type Left_Row_Indices   is (<>);
      type Right_Col_Indices  is (<>);
      type Right_Row_Indices  is (<>);
      type Output_Col_Indices is (<>);
      type Output_Row_Indices is (<>);
      type Left_Matrices is array (Left_Row_Indices,
                                   Left_Col_Indices) of Left_Elements;
      type Right_Matrices is array (Right_Row_Indices,
                                    Right_Col_Indices)
                                of Right_Elements;
      type Output_Matrices is array (Output_Row_Indices,
                                     Output_Col_Indices)
                                 of Output_Elements;
      with function "*" (Left  : Left_Elements;
                         Right : Right_Elements) return Output_Elements is <>;
      with function "+" (Left  : Output_Elements;
                         Right : Output_Elements) return Output_Elements is <>;
   package Matrix_Matrix_Multiply_Unrestricted is

      function "*" (Left  : Left_Matrices;
                    Right : Right_Matrices) return Output_Matrices;

   end Matrix_Matrix_Multiply_Unrestricted;

pragma PAGE:
   generic
      type Left_Elements      is digits <>;
      type Right_Elements     is digits <>;
      type Output_Elements    is digits <>;
      type M_Indices          is (<>);
      type N_Indices          is (<>);
      type P_Indices          is (<>);
      type Left_Matrices is array   (M_Indices, N_Indices) of Left_Elements;
      type Right_Matrices is array  (N_Indices, P_Indices) of Right_Elements;
      type Output_Matrices is array (M_Indices, P_Indices) of Output_Elements;
      with function "*" (Left  : Left_Elements;
                         Right : Right_Elements) return Output_Elements is <>;
      with function "+" (Left  : Output_Elements;
                         Right : Output_Elements) return Output_Elements is <>;
```

```
   function Matrix_Matrix_Multiply_Restricted
             (Left  : Left_Matrices;
              Right : Right_Matrices) return Output_Matrices;

pragma PAGE;
   generic
      type Left_Elements      is digits <>;
      type Right_Elements     is digits <>;
      type Output_Elements    is digits <>;
      type Left_Col_Indices   is (<>);
      type Left_Row_Indices   is (<>);
      type Right_Col_Indices  is (<>);
      type Right_Row_Indices  is (<>);
      type Output_Col_Indices is (<>);
      type Output_Row_Indices is (<>);
      type Left_Matrices is array (Left_Row_Indices,
                                   Left_Col_Indices) of Left_Elements;
      type Right_Matrices is array (Right_Row_Indices,
                                    Right_Col_Indices)
                                 of Right_Elements;
      type Output_Matrices is array (Output_Row_Indices,
                                     Output_Col_Indices)
                                  of Output_Elements;
      with function "*" (Left  : Left_Elements;
                         Right : Right_Elements) return Output_Elements is <>;
   package Matrix_Matrix_Transpose_Multiply_Unrestricted is

      function "*" (Left  : Left_Matrices;
                    Right : Right_Matrices) return Output_Matrices;

   end Matrix_Matrix_Transpose_Multiply_Unrestricted;

pragma PAGE;
   generic
      type Left_Elements      is digits <>;
      type Right_Elements     is digits <>;
      type Output_Elements    is digits <>;
      type M_Indices          is (<>);
      type N_Indices          is (<>);
      type P_Indices          is (<>);
      type Left_Matrices is array   (M_Indices, N_Indices) of Left_Elements;
      type Right_Matrices is array  (P_Indices, N_Indices) of Right_Elements;
      type Output_Matrices is array (M_Indices, P_Indices) of Output_Elements;
      with function "*" (Left  : Left_Elements;
                         Right : Right_Elements) return Output_Elements is <>;
   function Matrix_Matrix_Transpose_Multiply_Restricted
             (Left  : Left_Matrices;
              Right : Right_Matrices) return Output_Matrices;

pragma PAGE;
   generic
      type Left_Elements   is digits <>;
      type Right_Elements  is digits <>;
      type Result_Elements is digits <>;
      type Left_Indices    is (<>);
      type Right_Indices   is (<>);
      type Left_Vectors    is array (Left_Indices) of Left_Elements;
```

```
      type Right_Vectors   is array (Right_Indices) of Right_Elements;
      with function "*" (Left  : Left_Elements;
                         Right : Right_Elements) return Result_Elements is <>;
   package Dot_Product_Operations_Unrestricted is

      function Dot_Product (Left  : Left_Vectors;
                            Right : Right_Vectors) return Result_Elements;

   end Dot_Product_Operations_Unrestricted;

pragma PAGE;
   generic
      type Left_Elements   is digits <>;
      type Right_Elements  is digits <>;
      type Result_Elements is digits <>;
      type Indices         is (<>);
      type Left_Vectors    is array (Indices) of Left_Elements;
      type Right_Vectors   is array (Indices) of Right_Elements;
      with function "*" (Left  : Left_Elements;
                         Right : Right_Elements) return Result_Elements is <>;
   function Dot_Product_Operations_Restricted (Left  : Left_Vectors;
                                               Right : Right_Vectors)
                                              return Result_Elements;

pragma PAGE;
   generic
      type Elements                is digits <>;
      type Diagonal_Range          is range <>;
      type Full_Input_Col_Indices  is (<>);
      type Full_Input_Row_Indices  is (<>);
      type Full_Output_Col_Indices is (<>);
      type Full_Output_Row_Indices is (<>);
      type Diagonal_Matrices     is array (Diagonal_Range) of Elements;
      type Full_Input_Matrices   is array (Full_Input_Row_Indices,
                                           Full_Input_Col_Indices) of Elements;
      type Full_Output_Matrices is array (Full_Output_Row_Indices,
                                          Full_Output_Col_Indices) of Elements;
   package Diagonal_Full_Matrix_Add_Unrestricted is

      function "+" (D_Matrix : Diagonal_Matrices;
                    F_Matrix : Full_Input_Matrices) return Full_Output_Matrices;

   end Diagonal_Full_Matrix_Add_Unrestricted;

pragma PAGE;
   generic
      type Elements          is digits <>;
      type Diagonal_Range    is range <>;
      type Indices           is (<>);
      type Diagonal_Matrices is array (Diagonal_Range) of Elements;
      type Full_Matrices     is array (Indices, Indices) of Elements;
   function Diagonal_Full_Matrix_Add_Restricted
              (D_Matrix : Diagonal_Matrices;
               F_Matrix : Full_Matrices) return Full_Matrices;

pragma PAGE;
   generic
```

```
      type A_Elements    is digits <>;
      type B_Elements    is digits <>;
      type C_Elements    is digits <>;
      type M_Indices     is (<>);
      type N_Indices     is (<>);
      type A_Matrices is array( M_Indices, N_Indices ) of A_Elements;
      type B_Matrices is array( N_Indices, N_Indices ) of B_Elements;
      type C_Matrices is array( M_Indices, M_Indices ) of C_Elements;
      with function "*"( Left  : A_Elements;
                         Right : B_Elements ) return C_Elements is <>;
      with function "*"( Left  : C_Elements;
                         Right : A_Elements ) return C_Elements is <>;
   package Aba_Trans_Dynam_Sparse_Matrix_Sq_Matrix is

      function Aba_Transpose( A : A_Matrices;
                              B : B_Matrices )
                                return C_Matrices;

   end Aba_Trans_Dynam_Sparse_Matrix_Sq_Matrix;
pragma PAGE;
   generic
      type Vector_Elements        is digits <>;
      type Matrix_Elements        is digits <>;
      type Scalars                is digits <>;
      type Indices                is (<>);
      type Vectors is array( Indices ) of Vector_Elements;
      type Matrices is array( Indices, Indices ) of Matrix_Elements;
      with function "*"( Left  : Vector_Elements;
                         Right : Matrix_Elements )
                            return Vector_Elements is <>;
      with function "*"( Left  : Vector_Elements;
                         Right : Vector_Elements )
                            return Scalars is <>;
   package Aba_Trans_Vector_Sq_Matrix is
      function Aba_Transpose( A : Vectors;
                              B : Matrices ) return Scalars;
   end Aba_Trans_Vector_Sq_Matrix;


pragma PAGE;
   generic
      type Vector_Elements is digits <>;
      type Matrix_Elements is digits <>;
      type Scalars         is digits <>;
      type Indices         is (<>);
      type Vectors         is array( Indices ) of Vector_Elements;
      type Matrices        is array( Indices, Indices ) of Matrix_Elements;
      with function "*"( Left  : Vector_Elements;
                         Right : Scalars ) return Vector_Elements is <>;
      with function "*"( Left  : Vector_Elements;
                         Right : Vector_Elements ) return Matrix_Elements is <>;
   package Aba_Trans_Vector_Scalar is
      function Aba_Transpose( A : Vectors;
                              B : Scalars ) return Matrices;
   end Aba_Trans_Vector_Scalar;

pragma PAGE;
```

```
   generic

      type Vector_Elements is digits <>;
      type Indices is (<>);
      type Vectors is array( Indices ) of Vector_Elements;

   package Column_Matrix_Operations is

      type Column_Matrices is record
         Col_Vector     : Vectors;
         Diagonal       : BOOLEAN;
         Active_Column  : Indices;
      end record;

      function Set_Diagonal_And_Subtract_From_Identity
         ( Column        : Vectors;
           Active_Column : Indices ) return Column_Matrices;

      generic
         type B_Matrix_Elements is digits <>;
         type C_Matrix_Elements is digits <>;
         type B_Matrices is array( Indices, Indices ) of B_Matrix_Elements;
         type C_Matrices is array( Indices, Indices ) of C_Matrix_Elements;
         with function "*"( Left  : Vector_Elements;
                            Right : B_Matrix_Elements ) return B_Matrix_Elements is <>;
      function Aba_Transpose( A : Column_Matrices;
                              B : B_Matrices )   return C_Matrices;

      generic
         type B_Matrix_Elements is digits <>;
         type C_Matrix_Elements is digits <>;
         type B_Matrices is array( Indices, Indices ) of B_Matrix_Elements;
         type C_Matrices is array( Indices, Indices ) of C_Matrix_Elements;
         with function "*"( Left  : Vector_Elements;
                            Right : B_Matrix_Elements ) return B_Matrix_Elements is <>;
      function Aba_Symm_Transpose( A : Column_Matrices;
                                   B : B_Matrices )   return C_Matrices;

   end Column_Matrix_Operations;

end General_Vector_Matrix_Algebral;
```

3.6.8.3 STANDARD_TRIG TLCSC (CATALOG #P1-0)

This generic package provides a standard set of trigonometric functions.

The generic formal types allow the user to select the precision to be used for all calculations. Three types, radians, semicircles, and degrees, will be derived from the formal parameter 'angle'. These derived types will be used as inputs to the sine, cosine, and tangent subprograms and used as outputs from the arcsine, arccosine, and arctangent subprograms. Two types, sin_cos_ratio and tan_ratio, will be derived from the formal parameter 'trig_ratio'. These types will be used as inputs to the arcsine, arccosine, and arctangent subprograms and as outputs from the sine, cosine, and tangent subprograms.

3.6.8.3.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Type	Requirements Allocation
Sin	function	R086, R092, R098
Cos	function	R087, R093, R099
Sin_Cos	procedure	R086, R087, R092
		R093, R098, R099
Tan	function	R088, R094, R100
ArcSin	function	R089, R095, R101
ArcCos	function	R090, R096, R102
ArcSin_ArcCos	procedure	R089, R090, R095
		R096, R101, R102
ArcTan	function	R091, R097, R103
ArcTan2	function	

3.6.8.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Angle	floating point type	Used to determine precision of type Radians, Semicircles, and Degrees
Trig_Ratio	floating point type	Used to determine precision of type Sin_Cos_Ratio and Tan_Ratio

Data objects:

The following table summarizes the generic formal objects required by this part:

Name	Type	Value	Description
pi_value	Angle	N/A	Value to be used for pi

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following table describes the data types exported by this part:

Name	Base Type	Range	Description
Radians	Angle	-oo .. +oo	Radian unit of measurement
Semicircles	Angle	-oo .. +oo	Semicircle unit of measurement
Degrees	Angle	-oo .. +oo	Degree unit of measurement
Sin_Cos_Ratio	Trig_Ratio	-1 .. +1	Result of a sine or cosine function
Tan_Ratio	Trig_Ratio	-oo .. +oo	Result of a tangent function function

3.6.8.3.3 UTILIZATION OF OTHER ELEMENTS

At the package body level, this part with's the POLYNOMIALS package. This package contains packages of generic functions which provide various polynomial solutions to functions.

3.6.8.3.4 LOCAL ENTITIES

Subprograms:

A set of subprograms in the Polynomials package will need to be instantiated to satisfy the requirements of this part.

Packages:

A set of packages in the Polynomials package will need to be instantiated to satisfy the requirements of this part.

3.6.8.3.5 INTERRUPTS

None.

3.6.8.3.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Standard_Trig;
...
   type My_Angle      is digits 12;
   type My_Trig_Ratio is digits 12;
   ...
   My_Pi              : constant My_Angle := 3.141_592_653_5;
   ...
   package Trig is new Standard_Trig (Angle      => My_Angle,
                                      Trig_Ratio => My_Trig_Ratio,
                                      Pi_Value   => My_Pi);
   ...
   Angle_Value : Trig.Radians;
   Sin_Value   : Trig.Sin_Cos_Ratio;
   ...
   begin
      ...
      Sin_Value := Trig.Sin(Angle_Value);
```

3.6.8.3.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.3.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Input	Output
Sin	function	radians	sin_cos_ratio
Sin	"	semicircles	"
Sin	"	degrees	"
Cos	function	radians	sin_cos_ratio
Cos	"	semicircles	"
Cos	"	degrees	"
Sin_Cos	procedure	radians	SIN) sin_cos_ratio COS) "
Sin_Cos	"	semicircles	SIN) sin_cos_ratio COS) "
Sin_Cos	"	degrees	SIN) sin_cos_ratio COS) "
Tan	function	radians	tan_ratio
Tan	"	semisemicircles	"
Tan	"	degrees	"
ArcSin	function	sin_cos_ratio	radians
ArcSin	"	"	semicircles
ArcSin	"	"	degrees
ArcCos	function	sin_cos_ratio	radians
ArcCos	"	"	semicircles
ArcCos	"	"	degrees
ArcSin_ ArcCos	procedure	sin_cos_ratio	ASIN) radians ACOS) "
ArcSin_ ArcCos	"	"	ASIN) semicircles ACOS) "
ArcSin_ ArcCos	"	"	ASIN) degrees ACOS) "
ArcTan	function	tan_ratio	radians
ArcTan	"	"	semicircles
ArcTan	"	"	degrees
Arctan2	generic function	"	<angles>

The following table lists the catalog numbers for these parts:

Name	Type	Catalog #
Sin	function	P555-0
Sin	"	P556-0
Sin	"	P557-0
Cos	function	P558-0
Cos	"	P559-0
Cos	"	P560-0
Sin_Cos	procedure	P561-0
Sin_Cos	"	P562-0
Sin_Cos	"	P563-0
Tan	function	P564-0
Tan	"	P565-0
Tan	"	P566-0
ArcSin	function	P567-0
ArcSin	"	P568-0
ArcSin	"	P569-0
ArcCos	function	P570-0
ArcCos	"	P571-0
ArcCos	"	P572-0
ArcSin_ ArcCos	procedure	P573-0
ArcSin_ ArcCos	"	P574-0
ArcSin_ ArcCos	"	P575-0
ArcTan	function	P576-0
ArcTan	"	P577-0
ArcTan	"	P578-0

3.6.8.3.9 PART DESIGN

3.6.8.3.9.1 ARCTAN2 (FUNCTION SPECIFICATION) (CATALOG #P537-0)

This function calculates the arctangent of two input values defining the endpoint of a vector. The result of this function is the angle between the vector and the positive x-axis and is in the range equivalent to +/- pi.

If both X and Y equal 0, this function will return a value of 0.

3.6.8.3.9.1.1 REQUIREMENTS ALLOCATION

None

3.6.8.3.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Angles	floating point	Data type defining angular measurements
Measurements	floating point	Data type defining function input types

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Description
Cycle_over_2	Angles	Number of angular units of measurement in half a circle (e.g., pi/2 for Radians, 180 for Degrees, and 1.0 for Semicircles)
Cycle_over_4	Angles	Number of angular units of measurement in 1/4 of a circle (e.g., pi/4 for Radians, 90 for degrees, and .5 for Semicircles)

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"/"	function	Division operator defining the operation: Measurements / Measurements => Tan_Ratio
Arctan	function	Arctangent function

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
X	Measurements	in	First element of the coordinate pair
Y	Measurements	in	Second element of the coordinate pair

3.6.8.3.9.1.3 INTERRUPTS

None.

3.6.8.3.9.1.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Standard_Trig;
with Basic_Data_Types;
...
   ...
   package BDT renames Basic_Data_Types;
   use BDT;
   use Trig;
   ...
   function Atan2 is new BDT.Trig.Arctan2
               (Angles       => Radians,
                Measurements => Velocities,
                Cycle_over_2 => 3.14/2.0,
                Cycle_over_4 => 3.14/4.0);
   ...
   Angle           : Radians;
   Velocity_Vector : Velocity_Vectors := (X => ...,
                                          Y => ...,
                                          Z => 0.0);
   ...
   begin
      ...
      Angle := ATan2 (X => Velocity_Vector(X),
                      Y => Velocity_Vector(Y));
```

3.6.8.3.9.1.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.3.9.1.6 DECOMPOSITION

None.

(This page left intentionally blank.)

```
generic
   type Angle      is digits <>;
   type Trig_Ratio is digits <>;
   Pi_Value        : in Angle;
package Standard_Trig is

   type Radians     is new Angle;
   type Semicircles is new Angle;
   type Degrees     is new Angle;

   type Sin_Cos_Ratio is new Trig_Ratio range -1.0 .. 1.0;
   type Tan_Ratio     is new Trig_Ratio;

-- --Sine functions

   function Sin (Input : Radians)     return Sin_Cos_Ratio;  --Catalog #P555-0
   function Sin (Input : Semicircles) return Sin_Cos_Ratio;  --Catalog #P556-0
   function Sin (Input : Degrees)     return Sin_Cos_Ratio;  --Catalog #P557-0

-- --Cosine functions

   function Cos (Input : Radians)     return Sin_Cos_Ratio;  --Catalog #P558-0
   function Cos (Input : Semicircles) return Sin_Cos_Ratio;  --Catalog #P559-0
   function Cos (Input : Degrees)     return Sin_Cos_Ratio;  --Catalog #P560-0

-- --Sine-Cosine procedures

   procedure Sin_Cos (Input      : in  Radians;              --Catalog #P561-0
                      Sin_Result : out Sin_Cos_Ratio;
                      Cos_Result : out Sin_Cos_Ratio);
   procedure Sin_Cos (Input      : in  Semicircles;          --Catalog #P562-0
                      Sin_Result : out Sin_Cos_Ratio;
                      Cos_Result : out Sin_Cos_Ratio);
   procedure Sin_Cos (Input      : in  Degrees;              --Catalog #P563-0
                      Sin_Result : out Sin_Cos_Ratio;
                      Cos_Result : out Sin_Cos_Ratio);

-- --Tangent functions

   function Tan (Input : Radians)     return Tan_Ratio;      --Catalog #P564-0
   function Tan (Input : Semicircles) return Tan_Ratio;      --Catalog #P565-0
   function Tan (Input : Degrees)     return Tan_Ratio;      --Catalog #P566-0

-- --Arcsine functions

   function Arcsin (Input : Sin_Cos_Ratio) return Radians;     --Catalog #P567-0
   function Arcsin (Input : Sin_Cos_Ratio) return Semicircles; --Catalog #P568-0
   function Arcsin (Input : Sin_Cos_Ratio) return Degrees;     --Catalog #P569-0

-- --Arccosine functions

   function Arccos (Input : Sin_Cos_Ratio) return Radians;     --Catalog #P570-0
   function Arccos (Input : Sin_Cos_Ratio) return Semicircles; --Catalog #P571-0
   function Arccos (Input : Sin_Cos_Ratio) return Degrees;     --Catalog #P572-0

-- --Arcsine-Arccosine functions
```

```
   procedure Arcsin_Arccos (Input          : in   Sin_Cos_Ratio; --Catalog #P573-0
                            Arcsin_Result  : out  Radians;
                            Arccos_Result  : out  Radians);
   procedure Arcsin_Arccos (Input          : in   Sin_Cos_Ratio; --Catalog #P574-0
                            Arcsin_Result  : out  Semicircles;
                            Arccos_Result  : out  Semicircles);
   procedure Arcsin_Arccos (Input          : in   Sin_Cos_Ratio; --Catalog #P575-0
                            Arcsin_Result  : out  Degrees;
                            Arccos_Result  : out  Degrees);

-- --Arctangent functions

   function Arctan (Input : Tan_Ratio) return Radians;          --Catalog #P576-0
   function Arctan (Input : Tan_Ratio) return Semicircles;      --Catalog #P577-0
   function Arctan (Input : Tan_Ratio) return Degrees;          --Catalog #P578-0

pragma PAGE;
   generic
      type Angles       is digits <>;
      type Measurements is digits <>;
      Cycle_Over_2      : in Angles;
      Cycle_Over_4      : in Angles;
      with function "/" (Left  : Measurements;
                         Right : Measurements) return Tan_Ratio is <>;
      with function Arctan (Input : Tan_Ratio) return Angles    is <>;
   function Arctan2 (X : Measurements;
                     Y : Measurements) return Angles;

end Standard_Trig;
```

3.6.8.4 GEOMETRIC_OPERATIONS TLCSC (CATALOG #P113-0)

This part contains the CAMP routines which perform geometric functions relative to the Earth frame.

3.6.8.4.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Unit_Radial_Vector	R168
Unit_Normal_Vector	
Compute_Segment_and_Unit_Normal_Vector	R169
Compute_Segment_and_Unit_Normal_with_Arcsin	
Great_Circle_Arc_Length	R082

3.6.8.4.2 INPUT/OUTPUT

None.

3.6.8.4.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.8.4.4 LOCAL ENTITIES

None.

3.6.8.4.5 INTERRUPTS

None.

3.6.8.4.6 TIMING AND SEQUENCING

None.

3.6.8.4.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.4.8 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Unit_Radial_Vector	generic function	Computes the unit radial vector of a point
Unit_Normal_Vector	generic function	Computes the unit normal vector for a course segment
Compute_Segment_and_Unit_Normal_Vector	generic procedure	Computes the unit normal vector and arc length of a course segment using the assumption alpha=sin(alpha)
Compute_Segment_and_Unit_Normal_Vector_with_Arcsin	generic procedure	Computes the unit normal vector and arc length of a course segment NOT using the assumption alpha=sin(alpha)
Great_Circle_Arc_Length	generic package	Computes the great circle arc length between two points

3.6.8.4.9 PART DESIGN

3.6.8.4.9.1 UNIT_RADIAL_VECTOR (CATALOG #P114-0)

This part computes the unit radial vector of a point given the point's latitude and longitude. It extends outward from the origin of the Earth- centered reference frame towards the point whose latitude and longitude are given. The computations performed by this part are as follows:

```
UR(X) := Cos(Lat) * Cos(Long)
UR(Y) := Cos(Lat) * Sin(Long)
UR(Z) := Sin(Lat)
```

3.6.8.4.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R168.

3.6.8.4.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Indices	discrete type	Used to dimension arrays
Earth_Positions	floating point	Data type of longitude/latitude values
Sin_Cos_Ratio	floating point	Data type of results of sine/cosine routines
Unit_Vectors	array	One-dimensional, 3-element array of Sin_Cos_Ratio dimensioned by Indices

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
X	Indices	'FIRST	Index into first element of array
Y	Indices	'SUCC(X)	Index into second element of array
Z	Indices	'LAST	Index into third element of array

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Sin_Cos	procedure	Returns the sine and cosine of an input value

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Lat_of_Point	Earth_Positions	In	Latitude of point for which a unit radial vector is desired
Long_of_Point	Earth_Positions	In	Latitude of point for which a unit radial vector is desired

3.6.8.4.9.1.3 INTERRUPTS

None.

3.6.8.4.9.1.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Basic_Data_Types; use Basic_Data_Types;
with Geometric_Operations;
with Coordinate_Vector_Matrix_Algebra;
...
   package BDT  renames Basic_Data_Types;
   package Geo  renames Geometric_Operations;
   package CVMA renames Coordinate_Vector_Matrix_Algebra;
   ...
   type Indices is (X, Y, Z);
   ...
   package Unit_V_Opns is new CVMA.Vector_Operations ...
   subtype Unit_Vectors is Unit_V_Opns.Vectors;
   ...
   function U_Radial_Vector is new
               Geo.Unit_Radial_Vector
                  (Indices         => Indices,
                   Earth_Positions => BDT.Earth_Position_Radians,
                   Sin_Cos_Ratio   => BDT.Trig.Sin_Cos_Ratio,
                   Unit_Vectors    => Unit_Vectors,
                   Sin_Cos         => BDT.Trig.Sin_Cos);
   ...
   Lat      : BDT.Earth_Position_Radians;
   Long     : BDT.Earth_Positions_Radians;
   UR_A     : Unit_Vectors;
   ...
      begin
        ...
        UR_A := U_Radial_Vector
                   (Lat_of_Point  => Lat,
                    Long_of_Point => Long);
        ...
```

3.6.8.4.9.1.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.4.9.1.6 DECOMPOSITION

None.

3.6.8.4.9.2 UNIT_NORMAL_VECTOR (CATALOG #P115-0)

This function computes the segment unit normal vector for a course segment given the unit radial vectors for the two points defining the course segment. The computations performed by this part are as follows:

```
   UN_B := UR_B X UR_A / Length(UR_B X UR_A)
where UN_B :== unit normal vector
      UR_B :== unit radial vector to point B
      UR_A :== unit radial vector to point A
```

3.6.8.4.9.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R169.

3.6.8.4.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Unit_Vectors	private	One-dimensional, 3-element array of Sin_Cos_Ratio
Sin_Cos_Ratio	floating point type	Data type of results of sine/cosine routines

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"/"	function	Division operator defining the operation: Unit_Vectors / Sin_Cos_Ratio => Unit_Vectors
Cross_Product	function	Cross product function
Vector_Length	function	Vector length function

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_Radial_A	Unit_Vectors	In	Unit radial vector defining one endpoint of the course segment
Unit_Radial_B	Unit_Vectors	In	Unit radial vector defining one endpoint of the course segment

3.6.8.4.9.2.3 INTERRUPTS

None.

3.6.8.4.9.2.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Basic_Data_Types; use Basic_Data_Types;
with Geometric_Operations;
with Coordinate_Vector_Matrix_Algebra;
...
   package BDT  renames Basic_Data_Types;
   package Geo  renames Geometric_Operations;
   package CVMA renames Coordinate_Vector_Matrix_Algebra;
   ...
   type Indices is (X, Y, Z);
   ...
   package Unit_V_Opns is new CVMA.Vector_Operations ...
   subtype Unit_Vectors is Unit_V_Opns.Vectors;
   ...
   function My_Cross_Product is new CVMA.Cross_Product ...
   ...
   package Vector_Scalar_Opns is  new CVMA.Vector_Scalar_Operations ...
   ...
   function U_Normal_Vector is new
               Geo.Unit_Normal_Vector
                  (Unit_Vectors    => Unit_Vectors,
                   Sin_Cos_Ratio  => BDT.Trig.Sin_Cos_Ratio,
                   "/"            => Vector_Scalar_Opns."/",
                   Cross_Product  => My_Cross_Product,
                   Vector_Length  => Unit_V_Opns.Vector_Length);
   ...
   UR_A    : Unit_Vectors;
   UR_B    : Unit_Vectors;
   UN_B    : Unit_Vectors;
   ...
      begin
         ...
         UN_B := U_Normal_Vector
                   (Unit_Radial_A => UR_A,
                    Unit_Radial_B => UR_B);
         ...
```

3.6.8.4.9.2.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.4.9.2.6 DECOMPOSITION

None.

3.6.8.4.9.3 COMPUTE_SEGMENT_AND_UNIT_NORMAL_VECTOR (CATALOG #P116-0)

This procedure computes the segment unit normal vector for a course segment and the length of the course segment, given the unit radial vectors for the 2 points defining the course segment.
The computations performed by this part are as follows:

```
    UN_2     := UR_2 X UR_1 / Length(UR_2 X UR_1)
    Seg_Dist := Earth_Radius * Length(UR_2 X UR_1)
where UN_2     :== unit normal vector
      UR_2     :== unit radial vector to point 2
      UR_1     :== unit radial vector to point 1
      Seg_Dist :== great circle arc length between points 1 and 2
```

3.6.8.4.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R169.

3.6.8.4.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors; this type should have a length of 3
Earth_ Distances	floating point type	Data type used to define distance measurements involving the Earth's radius
Segment_ Distances	floating point type	Data type used to define distance measure- menets involving nagivation segments
Sin_Cos_Ratio	floating point type	Data type used to define results of sine/ cosine operations
Unit_Vectors	array	One-dimensional, 3-element array indexed by Indices and containing Sin_Cos_Ratio elements

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_Distances	N/A	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the operation: Earth_Distances * Sin_Cos_Ratio => Segment_Distances
"/"	function	Operator defining the operation: Unit_Vectors / Sin_Cos_Ratio => Unit_Vectors
Cross_Product	function	Calculates the cross product of two units
Vector_Length	function	Calculates the length of a vector

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_Radial1	Unit_Vectors	in	Unit radial vector to waypoint B
Unit_Radial2	Unit_Vectors	in	Unit radial vector to waypoint C
Unit_Normal2	Unit_Vectors	out	Segment unit normal vector
Segment_ Distance	Segment_ Distances	out	Great circle arc length between points 1 and 2

3.6.8.4.9.3.3 INTERRUPTS

None.

3.6.8.4.9.3.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Geometric_Parts;
with Basic_Data_Types; use Basic_Data_Types;
with Coordinate_Vector_Matrix_Algebra;
with WGS72_Ellipsoid_Engineering_Data;
...
   package Geo   renames Geometric_Parts;
   package BDT   renames Basic_Data_Types;
   package CVMA  renames Coordinate_Vector_Matrix_Algebra;
   package WGS72 renames WGS72_Ellipsoid_Engineering_Data;
   ...
   type Indices is (X, Y, Z);
   ...
   package Unit_V_Opns is new CVMA.Vector_Operations ...
   subtype Unit_Vectors is Unit_V_Opns.Vectors;
   ...
   package Vector_Scalar_Opns is new CVMA.Vector_Scalar_Operations ...
   ...
   function Cross_Prod is new CVMA.Cross_Product ...
   ...
   procedure Comp_Segment_and_U_N1_Vector is new
                Geo.Compute_Segment_and_Unit_Normal_Vector
```

```
                    (Indices                  => Indices,
                     Earth_Distances          => BDT.Meters,
                     Segment_Distances        => BDT.Meters,
                     Sin_Cos_Ratio            => BDT.Trig.Sin_Cos_Ratio,
                     Unit_Vectors             => Unit_Vectors,
                     Earth_Radius             => WGS72.Semimajor_Axis,
                     "/"                      => Vector_Scalar_Opns."/",
                     Cross_Product            => Cross_Prod,
                     Vector_Length            => Unit_V_Opns.Vector_Length);
   ...
   UR_B       : Unit_Vectors;
   UR_C       : Unit_Vectors;
   UN_C       : Unit_Vectors;
   BC_Dist    : BDT.Meters;
   ...
      begin
         ...
         BC_Dist := Comp_Segment_and_U_N1_Vector
                       (Unit_Radial_1     => UR_B,
                        Unit_Radial_2     => UR_C,
                        Unit_Normal_2     => UN_C,
                        Segment_Distance => BC_Dist);
         ...
```

3.6.8.4.9.3.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.4.9.3.6 DECOMPOSITION

None.

3.6.8.4.9.4 COMPUTE_SEGMENT_AND_UNIT_NORMAL_VECTOR_WITH_ARCSIN (CATALOG #P1049-0)

This procedure computes the segment unit normal vector for a course segment and the length of the course segment, given the unit radial vectors for the 2 points defining the course segment.
The computations performed by this part are as follows:

```
  UN_2     := UR_2 X UR_1 / Length(UR_2 X UR_1)
  Seg_Dist := Earth_Radius * Arcsin(Length(UR_2 X UR_1))
where UN_2     :== unit normal vector
      UR_2     :== unit radial vector to point 2
      UR_1     :== unit radial vector to point 1
      Seg_Dist :== great circle arc length between points 1 and 2
```

3.6.8.4.9.4.1 REQUIREMENTS ALLOCATION

None.

3.6.8.4.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors; this type should have a length of 3
Earth_ Distances	floating point type	Data type used to define distance measurements involving the Earth's radius
Segment_ Distances	floating point type	Data type used to define distance measure-menets involving nagivation segments
Sin_Cos_Ratio	floating point type	Data type used to define results of sine/ cosine operations
Unit_Vectors	array	One-dimensional, 3-element array indexed by Indices and containing Sin_Cos_Ratio elements

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_Distances	N/A	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the operation: Earth_Distances * Sin_Cos_Ratio => Segment_Distances
"/"	function	Operator defining the operation: Unit_Vectors / Sin_Cos_Ratio => Unit_Vectors
Arcsin	function	Calculates the arcsine of an input value
Cross_Product	function	Calculates the cross product of two units
Vector_Length	function	Calculates the length of a vector

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_Radial1	Unit_Vectors	in	Unit radial vector to waypoint B
Unit_Radial2	Unit_Vectors	in	Unit radial vector to waypoint C
Unit_Normal2	Unit_Vectors	out	Segment unit normal vector
Segment_ Distance	Segment_ Distances	out	Great circle arc length between points 1 and 2

3.6.8.4.9.4.3 INTERRUPTS

None.

3.6.8.4.9.4.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Geometric_Parts;
with Basic_Data_Types; use Basic_Data_Types;
with Coordinate_Vector_Matrix_Algebra;
with WGS72_Ellipsoid_Engineering_Data;
...
   package Geo    renames Geometric_Parts;
   package BDT    renames Basic_Data_Types;
   package CVMA   renames Coordinate_Vector_Matrix_Algebra;
   package WGS72  renames WGS72_Ellipsoid_Engineering_Data;
   ...
   type Indices is (X, Y, Z);
   ...
   package Unit_V_Opns is new CVMA.Vector_Operations ...
   subtype Unit_Vectors is Unit_V_Opns.Vectors;
   ...
   package Vector_Scalar_Opns is new CVMA.Vector_Scalar_Operations ...
   ...
   function Cross_Prod is new CVMA.Cross_Product ...
   ...
   procedure Comp_Segment_and_U_N1_Vector is new
                Geo.Compute_Segment_and_Unit_Normal_Vector_with_Arcsin
                  (Indices             => Indices,
                   Earth_Distances     => BDT.Meters,
                   Radians             => BDT.Trig.Radians,
                   Segment_Distances   => BDT.Meters,
                   Sin_Cos_Ratio       => BDT.Trig.Sin_Cos_Ratio,
                   Unit_Vectors        => Unit_Vectors,
                   Earth_Radius        => WGS72.Semimajor_Axis,
                   "/"                 => Vector_Scalar_Opns."/",
                   Cross_Product       => Cross_Prod,
                   Vector_Length       => Unit_V_Opns.Vector_Length);
   ...
   UR_B       : Unit_Vectors;
   UR_C       : Unit_Vectors;
   UN_C       : Unit_Vectors;
   BC_Dist    : BDT.Meters;
   ...
```

```
      begin
         ...
         BC_Dist := Comp_Segment_and_U_N1_Vector
                       (Unit_Radial_1      => UR_B,
                        Unit_Radial_2      => UR_C,
                        Unit_Normal_2      => UN_C,
                        Segment_Distance => BC_Dist);
         ...
```

3.6.8.4.9.4.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.4.9.4.6 DECOMPOSITION

None.

3.6.8.4.9.5 GREAT_CIRCLE_ARC_LENGTH (CATALOG #P117-0)

This package contains the function required to compute the great circle arc length of a course segment given the latitude and longitude of the two endpoints.

The circle arc length equals: circle radius * angle subtended by the arc
To define the angle subtended by the arc this part calculates the unit radial vectors to the end points of the arc. Since the radials vectors have a length of 1 and since it is assumed the angle subtended by the arc is "relatively small", the following is true:

```
arc length := radius * angle
           := radius * sin(angle)
           := radius * (1)*(1)*sin(angle)
           := radius * (length of UR_A)*(length of UR_B) * sin(angle)
           := radius * length(URB X UR_A)
```

3.6.8.4.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirements R082.

3.6.8.4.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Earth_ Distances	floating point type	Data type used to define distance measurements involving the Earth's radius
Segment_ Distances	floating point type	Data type used to define distance measure- menets involving nagivation segments
Earth_Positions	floating point type	Data type of longitude/latitude measurements
Sin_Cos_Ratio	floating point type	Data type of results of sine/cosine routines

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_Distances	N/A	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the operation: Earth_Distances * Sin_Cos_Ratio => Segment_Distances
Sqrt	function	Square root function
Sin_Cos	procedure	Returns the sine and cosine of an input value

3.6.8.4.9.5.3 LOCAL ENTITIES

None.

3.6.8.4.9.5.4 INTERRUPTS

None.

3.6.8.4.9.5.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Basic_Data_Types; use Basic_Data_Types;
with General_Purpose_Math;
with Geometric_Operations;
with WGS72_Ellipsoid_Engineering_Data;
```

```
...
   package BDT    renames Basic_Data_Types;
   package GPMath renames General_Purpose_Math;
   package Geo    renames Geometric_Operations;
   package WGS72  renames WGS72_Ellipsoid_Engineering_Data;
   ...
   package Sq_Rt is new GPMath.Square_Root ...
   ...
   package GC_Arc_Length is new
              Geo.Great_Circle_Arc_Length
                 (Earth_Distances          => BDT.Meters,
                  Segment_Distances        => BDT.Meters,
                  Earth_Positions          => BDT.Earth_Position_Meters,
                  Sin_Cos_Ratio            => BDT.Trig.Sin_Cos_Ratio,
                  Earth_Radius             => WGS72.Semimajor_Axix,
                  Sqrt                     => Sq_Rt.Sqrt,
                  Sin_Cos                  => BDT.Trig.Sin_Cos);
   ...
   Lat_A      : BDT.Earth_Position_Radians;
   Lat_B      : BDT.Earth_Position_Radians;
   Long_A     : BDT.Earth_Position_Radians;
   Long_B     : BDT.Earth_Position_Radians;
   Arc_Length : BDT.Meters;
   ...
      begin
         ...
         Arc_Length
:= GC_Arc_Length.Compute
                           (Latitude_A  => Lat_A,
                            Latitude_B  => Lat_B,
                            Longitude_A => Long_A,
                            Longitude_B => Long_B);
         ...
```

3.6.8.4.9.5.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.4.9.5.7 DECOMPOSITION

The following table describes the decomposition of this part:

Name	Type	Description
Compute	function	Computes the great circle arc length

3.6.8.4.9.5.8 PART DESIGN

None.

```
package Geometric_Operations is

pragma PAGE;
   generic
      type Indices         is (<>);
      type Earth_Positions is digits <>;
      type Sin_Cos_Ratio   is digits <>;
      type Unit_Vectors    is array (Indices) of Sin_Cos_Ratio;
      X : in Indices       := Indices'FIRST;
      Y : in Indices       := Indices'SUCC(X);
      Z : in Indices       := Indices'LAST;
      with procedure Sin_Cos (Input  : in  Earth_Positions;
                              Sine   : out Sin_Cos_Ratio;
                              Cosine : out Sin_Cos_Ratio) is <>;
   function Unit_Radial_Vector
               (Lat_Of_Point  : Earth_Positions;
                Long_Of_Point : Earth_Positions) return Unit_Vectors;

pragma PAGE;
   generic
      type Unit_Vectors  is private;
      type Sin_Cos_Ratio is digits <>;
      with function "/" (Left  : Unit_Vectors;
                         Right : Sin_Cos_Ratio) return Unit_Vectors is <>;
      with function Cross_Product (Left  : Unit_Vectors;
                                   Right : Unit_Vectors)
                                  return Unit_Vectors is <>;
      with function Vector_Length (Input : Unit_Vectors)
                                  return Sin_Cos_Ratio is <>;
   function Unit_Normal_Vector
               (Unit_Radial_A : Unit_Vectors;
                Unit_Radial_B : Unit_Vectors) return Unit_Vectors;

pragma PAGE;
   generic
      type Indices           is (<>);
      type Earth_Distances   is digits <>;
      type Segment_Distances is digits <>;
      type Sin_Cos_Ratio     is digits <>;
      type Unit_Vectors      is array (Indices) of Sin_Cos_Ratio;
      Earth_Radius           : in Earth_Distances;
      with function "*" (Left  : Earth_Distances;
                         Right : Sin_Cos_Ratio) return Segment_Distances is <>;
      with function "/" (Left  : Unit_Vectors;
                         Right : Sin_Cos_Ratio) return Unit_Vectors is <>;
      with function Cross_Product (Left  : Unit_Vectors;
                                   Right : Unit_Vectors)
                                  return Unit_Vectors is <>;
      with function Vector_Length (Input : Unit_Vectors)
                                  return Sin_Cos_Ratio is <>;
   procedure Compute_Segment_And_Unit_Normal_Vector
               (Unit_Radial1     : in     Unit_Vectors;
                Unit_Radial2     : in     Unit_Vectors;
                Unit_Normal2     :    out Unit_Vectors;
                Segment_Distance :    out Segment_Distances);

pragma PAGE;
```

```
   generic
      type Indices            is (<>);
      type Earth_Distances    is digits <>;
      type Radians            is digits <>;
      type Segment_Distances is digits <>;
      type Sin_Cos_Ratio      is digits <>;
      type Unit_Vectors       is array (Indices) of Sin_Cos_Ratio;
      Earth_Radius            : in Earth_Distances;
      with function "*" (Left  : Earth_Distances;
                         Right : Radians) return Segment_Distances is <>;
      with function "/" (Left  : Unit_Vectors;
                         Right : Sin_Cos_Ratio) return Unit_Vectors is <>;
      with function Arcsin (Input : Sin_Cos_Ratio) return Radians is <>;
      with function Cross_Product (Left  : Unit_Vectors;
                                   Right : Unit_Vectors)
                                  return Unit_Vectors is <>;
      with function Vector_Length (Input : Unit_Vectors)
                                  return Sin_Cos_Ratio is <>;
   procedure Compute_Segment_And_Unit_Normal_Vector_With_Arcsin
               (Unit_Radial1      : in      Unit_Vectors;
                Unit_Radial2      : in      Unit_Vectors;
                Unit_Normal2      :     out Unit_Vectors;
                Segment_Distance :     out Segment_Distances);

pragma PAGE;
   generic
      type Earth_Distances    is digits <>;
      type Earth_Positions    is digits <>;
      type Segment_Distances is digits <>;
      type Sin_Cos_Ratio      is digits <>;
      Earth_Radius            : in Earth_Distances;
      with function "*" (Left  : Earth_Distances;
                         Right : Sin_Cos_Ratio) return Segment_Distances is <>;
      with function Sqrt (Input  : Sin_Cos_Ratio) return Sin_Cos_Ratio is <>;
      with procedure Sin_Cos (Input  : in  Earth_Positions;
                              Sine   : out Sin_Cos_Ratio;
                              Cosine : out Sin_Cos_Ratio) is <>;
   package Great_Circle_Arc_Length is

      function Compute (Latitude_A  : Earth_Positions;
                        Latitude_B  : Earth_Positions;
                        Longitude_A : Earth_Positions;
                        Longitude_B : Earth_Positions) return Segment_Distances;

   end Great_Circle_Arc_Length;

end Geometric_Operations;
```

SUPPLEMENTARY

INFORMATION

DEPARTMENT OF THE AIR FORCE
WRIGHT LABORATORY (AFSC)
EGLIN AIR FORCE BASE, FLORIDA, 32542-5434

ERRATA

AD-B120 253

REPLY TO ATTN OF: MNOI

13 Feb 92

SUBJECT: Removal of Distribution Statement and Export-Control Warning Notices

TO: Defense Technical Information Center
ATTN: DTIC/HAR (Mr William Bush)
Bldg 5, Cameron Station
Alexandria, VA 22304-6145

1. The following technical reports have been approved for public release by the local Public Affairs Office (copy attached).

Technical Report Number	AD Number
1. 88-18-Vol-4	ADB 120 251
2. 88-18-Vol-5	ADB 120 252
3. 88-18-Vol-6	ADB 120 253
4. 88-25-Vol-1	ADB 120 309
5. 88-25-Vol-2	ADB 120 310
6. 88-62-Vol-1	ADB 129 568
7. 88-62-Vol-2	ADB 129 569
8. 88-62-Vol-3	ADB 129-570
9. 85-93-Vol-1	ADB 102-654
10. 85-93-Vol-2	ADB 102-655
11. 85-93-Vol-3	ADB 102-656
12. 88-18-Vol-1	ADB 120 248
13. 88-18-Vol-2	ADB 120 249
14. 88-18-Vol-7	ADB 120 254
15. 88-18-Vol-8	ADB 120 255
16. 88-18-Vol-9	ADB 120 256
17. 88-18-Vol-10	ADB 120 257 *
18. 88-18-Vol-11	ADB 120 258
19. 88-18-Vol-12	ADB 120 259

2. If you have any questions regarding this request call me at DSN 872-4620.

Lynn S. Wargo

LYNN S. WARGO
Chief, Scientific and Technical
Information Branch

1 Atch
AFDTC/PA Ltr, dtd 30 Jan 92

ERRATA

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE DEVELOPMENT TEST CENTER (AFSC)
EGLIN AIR FORCE BASE, FLORIDA 32542-5000

REPLY TO ATTN OF: PA (Jim Swinson, 882-3931)

30 January 1992

SUBJECT: Clearance for Public Release

TO: WL/MNA

The following technical reports have been reviewed and are approved for public release: AFATL-TR-88-18 (Volumes 1 & 2), AFATL-TR-88-18 (Volumes 4 thru 12), AFATL-TR-88-25 (Volumes 1 & 2), AFATL-TR-88-62 (Volumes 1 thru 3) and AFATL-TR-85-93 (Volumes 1 thru 3).

Virginia N. Pribyla

VIRGINIA N. PRIBYLA, Lt Col, USAF
Chief of Public Affairs

AFDTC/PA 92-039